Honeywell

# BATCH OPERATING SYSTEM

SERIES 16

SOFTWARE

Doc. No. 41286103142F

CONTENTS

## NOTE

In this manual, the '$' character is referred to. Some teletype terminals used in the United Kingdom have both the '£' and '$' characters on the keyboard.

Reference in this manual to '£' character always means 'Shift 3'. On nearly all teletypes the character '£' is printed as #. The '$' character always means 'Shift 4'.

# 1 INTRODUCTION

The Honeywell Series 16 Batch Operating System (BOS) is a disc based batch operating system which controls functions such as assembly, compilation, loading and execution of programs.

## 1.1 General

The speed and expense of modern computers demand an efficient relationship between the computer and its operating procedures. In any installation an attempt should be made to satisfy economically the following requirements:

(1) Efficient utilisation of machine time.

(2) Streamline running of programs and minimise time wasted between jobs.

(3) Standardise operating techniques so that the running of any one program follows a general pattern.

(4) Inform operator of given requirements and/or situations during the processing of a program.

(5) Make available to all users: all subsystems required in a given environment and a system library applicable to environmental requirements.

Clearly, it will be necessary for the given operating system to use a minimal amount of memory for its own core resident routine with the option of self-expansion with a transitional routine read into memory for house-keeping purposes between the execution of control commands. Additionally, flexibility over the choice of subsystems is desirable in order to meet the needs of all possible environments. Honeywell have developed this operating system to meet the requirements outlined above.

## 1.2 Applicable Documents

Programmer's Reference Manual

FORTRAN Translator Manual

DAP-16 and DAP-16 Mod. 2 Manual

EDIT 700 Manual

Debug Listing

NOTE: For document numbers of the above manuals, please refer to Appendix F.

## 1.3 Functional Description

BOS is a comprehensive batch-oriented system for controlling assemblies, compilations, loading and execution of programs. BOS is available to users of Honeywell Models 316, 516, and 716 General Purpose Digital Computers, and offers three distinct advantages:

(1) Efficient system operation with minimum operator intervention that increases operational reliability.

(2) Batch processing capability that reduces throughput time.

(3) A system of diagnostic messages.

BOS is an operating system which facilitates the processing of jobs sequentially in batches. BOS allows batch processing to proceed without the operator having to set up processing parameters or select input/output devices. Operating in this mode greatly increases the utilisation of the computer system by efficiently handling the execution and transfer on completion of the previous job to the next job in the batch. The user directs and controls BOS through the use of Control Commands. The user utilises the various Control Commands to describe the job to the operating system. Thus, preceding the program with the appropriate Control Commands permits intermixing and uninterrupted processing of assemblies, compilations and executions. The Control Commands direct the construction and execution of programs and provide the link between the program and its environment. The environment includes BOS, its processors (including DAP-16 and FORTRAN), utility programs, the computer user and various peripheral input/output devices. A detailed dictionary of *Control Commands* is given in Section 3.

Printouts of control and error messages are made available during processing, and the operator is concerned only with the setting up of tapes, loading of cards, etc. If a program fails, the operator inspects the hard copy of control information and makes the necessary adjustments in input/output assignments, tape designations, etc. BOS is disc orientated and uses a defined area of the disc file for its own 'backing store'. All system and user files are defined within this area. Access to the various files is made through the file handling facilities available in BOS. Any disc space outside the area defined for use by BOS may be used freely by the user for his own purposes.

## 1.4 BOS System Hardware Configuration

The minimum hardware requirement for BOS is:

(1) A Honeywell Model 316, Model 516 or Model 716 computer (possessing between 12K and 32K words of core memory)

(2) A Fixed or Moving Head Disc Drive (possessing a minimum of 120K words of storage)

(3) A Console Typewriter (Teletype ASR-33/35)

## 1.5 Special Instructions

The following convention is adopted throughout this manual: A number in octal code is denoted by an apostrophe placed in front of the number, e.g., '000020.

# 2 BOS SYSTEMS

BOS main systems can be made in several different versions, depending upon the equipment configuration in which BOS is to function, namely, the type of backing store employed (Moving-or Fixed-Head Disc Drives) and the amount of core store available (minimum 12K).

A library and sub-systems are available, all of which supplement the main system. The various sub-systems are detailed in Appendix B. They include the following major sub-systems.

| | |
|---|---|
| DAP-16 Assembler | $DA |
| Debug | $DE |
| FORTRAN Translator | $FN |
| Relocating Loader | $LO |
| Source Editing Program | $ED |

Appendix B gives a full list of all available sub-systems, and a brief description of each.

## 2.1 Operating Environments

The operating environment in which this system is to function is given below. The appropriate input/output facilities are assumed. BOS requires for its use the following minimum configuration of computer equipment.

(1) A Honeywell Model 316, 516 or 716 General Purpose Digital Computer, processing a minimum of 12288 words of core storage.

(2) A backing store which may be either a fixed of moving-head disc drive.

A defined area of the disc file is used as the storage medium for BOS.

All major elements of BOS are stored on the disc in such a way that they are always available to the system.

(3) A console typewriter ASR/KSR to be used by the operating system to communicate with the computer operator.

(4) One or more of the following:

Up to four magnetic tape transports on one controller.

A Card Reader or a Card Reader/Punch

A Buffered line printer.

A high speed paper tape reader and punch

## 2.2 How BOS Performs its Functions

BOS performs its functions between jobs and does not exercise control over the execution of a program once that program has been loaded and control has been transferred to it. The functions are indicated to BOS via Control Commands.

Upon request, BOS loads a program and then relinquishes control of the computer and its associated peripheral equipment to the program. The only possible way BOS can regain control of the computer is if the BOS main system is reloaded. This may be done manually by the console operator or under program control by the program being executed.

## 2.3 Functions of BOS

BOS is an off-line Batch Operating System, which through the medium of the appropriate control commands, provides certain processing functions. Some of the more commonly required functions and uses of BOS are

(1) Clear all user files from the backing store.

(2) Establish a source, object or binary file on the backing store from any suitable input device.

(3) Delete a particular source, object or binary file from the backing store.

(4) List a source file on either the console typewriter (ASR in print mode) or the line printer, with headings and page numbers as appropriate.

(5) Punch a source or object file on the high-speed paper-tape punch or card punch.

(6) Verify two files on the backing store against one another.

(7) Update a source file on the backing store, using an update file on the backing store to produce a new source file, and print file on the backing store.

(8) Compile a FORTRAN source file from the backing store.

(9) Assemble a DAP-16 source file from the backing store.

(10) Link and load one or more object files from the backing store into the core store and either execute immediately or store into a new file for later execution.

## 2.4 BOS Storage

Storage of files on the backing store devices is allocated automatically by the operating system. BOS allocates storage in a manner which achieves the best utilisation of memory on the backing store. It accomplishes this by reading and writing in fixed length blocks. The size of these blocks is determined when the system is configured. (See Section 5.1.) The blocks are subsequently chained together to form files.

To allocate storage, the disc unit is divided up into two areas at system generation time; one area for system files, the other area for user files. The file directories specify the storage allocation for particular files. It is also possible at system generation time to specify an area of disc file not accessible to BOS. This area is utilised for special user programs.

## 2.4.1 File Names

Files have names of up to 6 characters in length. They may consist of any combination of letters or figures, but the first character must be either a letter or a '$' sign. All files that have names starting with a '$' are system files.

The system files may include all standard software: the DAP-16 assembler, the FORTRAN Translator and mathematical and input/output library routines. The user may create additional system files simply by naming these files with a label of the form $ followed by up to five characters. More than 5 characters are allowed, but only the first 5 are significant and must be letters or digits 0-9.

# 3 CONTROL COMMANDS

BOS offers complete and easy-to-use functions which relieve the user of many tedious handling tasks. Jobs are run by specifying a job description, through the medium of control commands, which are subsequently input through a command input device. This input device may be either the console keyboard (ASR), a paper tape reader, a card reader, a magnetic tape deck or the disc itself. The operator specifies the input/output devices which are required to process the current job. For example, for a FORTRAN compilation, the source input, object output and listing output need to be specified. This is achieved by the ATTACH control command, which causes a particular physical input/output device to be attached to an input/output stream. Devices remain attached until another ATTACH command is issued. Stream names are assigned two-letter codes; CI for the command input, SO for source output, etc. Similarly, input/output devices are given two-letter codes; PR for paper tape reader, AP for console typewriter (ASR in print mode), CR for card reader, etc. The stream and device names are given in Tables 3 - 1 and 3 - 2.

TABLE 3 - 1
STREAM NAMES

| Stream Name | Description | Input/Output | Mode | Default Device |
|---|---|---|---|---|
| CI | Command Input | Input | Source | ASR keyboard |
| SI | Source Input | Input | Source | Paper tape reader |
| OI | Object Input | Input | Object | Paper tape reader |
| BI | Binary Input | Input | Binary | Paper tape reader |
| EO | Error Output | Output | Source | ASR printer |
| SO | Source Output | Output | Source | ASR printer |
| OO | Object Output | Output | Object | Paper tape punch |
| BO | Binary Output | Output | Binary | Paper tape punch |

TABLE 3 - 2
DEVICE NAMES

| Device Name | Description | Input/Output | Modes |
|---|---|---|---|
| PR | Paper tape reader | Input | Source, object or binary |
| CR | Card reader | Input | Source, object or binary |
| Mn | Magnetic tape physical unit no. n (n = 0, 1, 2 or 3) | Either | Source, object or binary |
| AK | ASR keyboard | Input | Source |
| PP | Paper tape punch | Output | Source, object or binary |
| LP | Line printer | Output | Source |
| AP | ASR printer | Output | Source |
| DI * | Disc drive | Input | Source |
| CP | Card punch | Output | Source, object or binary |

* See section 9 for information on the use of the disc as a source input device

To create a new file on the backing store, the user must give an ESTABLISH command, which specifies a file name and whether the information in this file is in source, object or binary format. Source information is terminated by a source end-of-file record or the next control command, whereas object information must be terminated by an object end-of-file record only.

The various control commands are listed and described (see Sections 3.1 to 3.47). For purposes of illustration, the format of the control commands is depected with the aid of punched cards. In practice, the control commands could equally well be input through a punched paper tape, the console keyboard (ASR), a magnetic tape deck or the disc itself. It should be noted that only the first two characters after the dollar sign ($) are significant; subsequent characters may be omitted so as to save time and space.

Note: The command format for the Series 16 Batch Operating System is as follows:

1. *A dollar sign in column 1.*
2. *The command mnemonic in columns 2 onwards. As many letters as required may be inserted as long as there are at least 2 and none are spaces.*
3. *A space.*
4. *A parameter if required. This may be an identifier which begins with a letter or dollar sign or asterisk and followed by any number of letters or digits, or an octal constant which consists of up to 8 digits and uses only characters 0 to 7.*
5. *If another parameter is required it must be preceded by a comma or equal sign. The sequence is then repeated from item 4, e.g.,* $FO $SOURCE,OBJ,L,10
6. *If column 1 of the command record does not contain a $ sign the entire record is interpreted as a comment and is ignored.*

## 3.1 ABORT

$ABORT

This command sets the system error marker which causes the current job to be aborted, that is, the command input device is scanned until a $RESET, or $JOB command is input. Only the following commands are obeyed in ABORT mode: $ATTACH, $MREWIND, $DO, $WAIT, $JOB and $RESET. The commands $NEXT and $TYPE cause the next record input on the command input stream to be skipped but do not output the record. Should control be lost (i.e., a program goes into an infinite loop), a manual abort operation may be performed to go on to the next job. For details of this operation refer to Section 6.

Note: This command has no effect when input whilst Command Input is attached to the ASR Keyboard.

## 3.2 ANNOTATE

$ANNOTATE <OLDMASTER>,<NEW MASTER>

This command calls in the annotate subsystem, $AN, to annotate an existing DAP-16 source file or old master and produce a new file called the new master. Note that only two files are involved.

The subsystem makes two passes through the old master file. During the first pass it builds up a dictionary, in core, of labels from columns 1 to 4, and corresponding comments from columns 30 to 72. Lines with an asterisk or space in column 1 are ignored during this pass. During the second pass, the following are copied unchanged from the old master to the new master:

1. *End-of-group records.*
2. *Object records.*
3. *Special records (e.g., those produced by unformatted WRITE instructions).*
4. *Source records with an asterisk in column 1.*
5. *Source records with a space in column 12.*
6. *Source records not containing a space in column 30.*

Records possessing the characters 'ANN' in columns 6 to 8 are ignored during the second pass. These records may be used for adding entries to the dictionary of labels and comments.

If the input record is not one of the above six types,and the characters in columns 12 to 15 of the address field agree exactly with an entry in the dictionary of labels and comment, then provided that the comments field (column 30) is also blank, the comment from the dictionary is copied into columns 30 to 72. If no entry is found in the dictionary the record is output unchanged.

As an example, suppose that the old master source file contains the following:

```
            . . .
            STA        C
            STA        IM

            . . .
NR          ANN                    NEXT RECORD
NR          LDA        IM

            . . .
            JST        ERRR

            . . .
            JST        ERRR

            . . .
            IRS        C
            JMP        NR
            LDA        =1
            STA        IM

            . . .
ERRR        DAC        **          ERROR ROUTINE

            . . .
            JMP*       ERRR        EXIT

            . . .
C           BSZ        1           COUNTER
IM          BSZ        1           INPUT MARKER
=1          ANN                    TRUE
            END                    END OF ANNEX
```

Then the following information will be recorded in the new master file:

```
                . . .
                STA        C          COUNTER
                STA        IM         INPUT MARKER

                . . .
NR              LDA        IM         INPUT MARKER

                . . .
                JST        ERRR       ERROR ROUTINE

                . . .
                JST        ERRR       ERROR ROUTINE

                . . .
                IRS        C          COUNTER
                JMP        NR         NEXT RECORD
                LDA        =1         TRUE
                STA        IM         INPUT MARKER

                . . .
ERRR            DAC        **         ERROR ROUTINE

                . . .
                JMP*       ERRR       EXIT

                . . .
C               BSZ        1          COUNTER
IM              BSZ        1          INPUT MARKER
                END                   END OF ANNEX
```

There is a limit to the number of entries in the dictionary of labels and comments. Should this limit be exceeded a system error (MO) is generated. (Full details of system errors are given in Appendix A.) For core sizes of 8K, 12K and 16K this limit is 224, 395 and 566 labels respectively.

## 3.3 ATTACH

$ATTACH <stream name>,<device name>

This command causes a particular physical input/output device to be attached to an input/output stream. As an example, the following command connects the source input stream to the card reader. Thus, all subsequent calls for source input will cause the card reader to be used.

$ATTACH SI,CR

Attach commands may occur before, at the beginning of, or within the content of any job. ATTACH commands are usually supplied by the operator rather than by the programmer and they are executed even if the current job is being aborted. Input/output devices remain attached until another ATTACH command is issued. If no ATTACH command for a particular stream is supplied before the stream is used, a default device is used. The standard default devices are shown in Table 3 - 1.

## 3.4 BACK

$BACK

The effect of this command is to cause control to revert to the command following the last $CALL command. It should be noted that it is not advisable to execute a $DO command between a $CALL command and a $BACK command.

## 3.5 BDISC

$BDISC <n>

This command means move backwards in the command file by the specified octal number of records as denoted by the octal number n. Thus, the command pointer is decremented by the specified octal number n. At the start of and during the execution of a command in a disc file, the command pointer always points to the next command in sequence. Therefore, the command $BD 0 has no effect and the command $BD 1 causes a permanent loop on this command which is trapped. This causes a system error.

## 3.6 CALL

$CALL <file name>,<optional octal constant>

The effect of this command is to cause the BOS commands contained in the named file on the disc to be executed. The comma and the octal constant, which may consist of up to eight digits, are optional. If this option is included, the specified octal constant is inserted into the senselight word.

The named file is transferred to a scratch area on the disc and the BOS command input is subsequently attached to the start of the scratch area. For ease of handling commands, certain rules must be obeyed when constructing the named file. These rules are listed as follows:

(1) All commands must begin with the character percent (%) which is translated to dollar ($) when the file is transferred to the scratch area.

(2) All records in a command file must be less than or equal to decimal 22 characters in length.

(3) The number of records in a file is configurable (see Subsection 9.2 and Appendix E).

(4) Control can be returned to the calling file by the use of a %BACK command. The named file must not contain a %CALL. Control cannot be returned to the calling file if the named file contains a %DO command.

## 3.7 COMMAND

$COMMAND <file name>,<. . . .>

The effect of this command is to cause a buffer area to be generated in core with space for 120 characters. Execution of this command causes a dollar ($) character to be loaded into column 1. The first of the named files is then accessed and the first character string up to and including the first space is copied into the buffer. The next and any subsequent files are accessed in turn. In each case, the first character string up to and excluding the first space is copied into the buffer and is followed by a comma. This process is repeated until all of the specified files have been accessed or until the buffer becomes full. Any trailing comma is suppressed. The character string which has been compiled in the buffer is then obeyed as a command. If the first character of any of the accessed files is a space, the space will be ignored and the file will be copied into the buffer up to the first space from columns 2-120. For example, the following commands cause the program called SORT to be executed with octal parameters 1324 and 5777 set into high core.

```
$ES  EX,SI
EX
$ES  NA,SI
SORT
$ES  OC,SI
1324,5777
$CO  EX,NA,OC
```

The command as executed would be: $EX SORT,1324,5777

## 3.8 DAP

$DAP <source>, <object>, <optional listing>, <conditional statement listing option>

The source file is to be assembled in two-pass mode, in order to produce an object file if specified. Any end-of-group records occurring within the source file are ignored. The fourth parameter, normally omitted, controls the listing of conditional statements whose assembly is inhibited. If this parameter is either zero or omitted the listing of such statements is inhibited: if the parameter is a '1' the listing of these statements is forced. Clearly this parameter has no meaning if the listing file is not specified.

The object file may subsequently be loaded by a $LOAD command. An errors-only listing may be obtained by outputting the listing file to the error output stream, or a full listing by outputting it to the source output stream, for example:

```
$DAP    S, O, L2
$OUTPUT   L2, EO
$OUTPUT   L2, SO
```

An end-of-group record is written out after every individual DAP-16 program module, to both the object and listing files.

## 3.9 DELETE

$DELETE <file name 1>,<file name 2>,<. . . .>

This causes the named file(s) to be deleted from the backing store, thereby releasing space for other files.

The DELETE command can be used to delete either system or user file. It is the only command which enables a system file to be deleted. (See also $JOB command, Section 3.20.)

## 3.10 DICTIONARY

$DICTIONARY <file name> $\begin{bmatrix} ,U \\ ,S \end{bmatrix}$

This command establishes a file containing a dictionary of the files currently held on the disc. The optional parameter 'U' or 'S' may be given to restrict the information provided to either 'User' or 'System' files respectively. If neither 'U' nor 'S' is specified, details of both user and system files will be contained in the output file. For each file on the disc, the output file will contain the following information in source record format:

(1) 6 character file name

(2) File type – BINARY
SOURCE
OBJECT
SPECIAL
EMPTY

(3) For BINARY files, the START, LOW and HIGH addresses.

(4) The number of disc blocks used to hold the file.

(5) The number of groups within the file.

(6) The number of records within the file.

In addition, the total number of blocks used by the System and User Libraries is given.

The file established may be output on any source output device.

## 3.11 DO

$DO <file name>,<optional octal constant>

The command causes the BOS Commands contained in the named file on the disc to be executed. The comma and the octal constant, which may consist of up to eight digits, are optional. If this option is included, the specified constant is inserted into the senselight word.

The named file is transferred to a scratch area on the disc and the BOS command input is subsequently attached to the start of that scratch area. This command is obeyed even when the system error marker is set. For ease of handling commands, certain rules must be obeyed when constructing the named file. These rules are listed as follows:

(1) All commands must begin with the character percent (%) which is translated to dollar ($) when the file is transferred to the scratch area.

(2) All records in a command file must be less than or equal to decimal 22 characters in length

(3) The number of records in a file is configurable (see Subsection 9.2 and Appendix E).

Section 9 gives further information on the use of command files.

## 3.12 END

$END

This command has no effect as a control command. If it is input while a source file is being ESTABLISHed (refer to Section 3.14), it causes an end-of-file record to be written and the file to be closed. Any parameters present in the file are ignored.

## 3.13 ERRORS

$ERRORS <file name>

The effect of this command is to cause the named file to be scanned for error records. If any error records are found, the next record input on the Command Input stream, if a command, is obeyed. If no error records are found the next record input on the Command Input stream is skipped.

## 3.14 ESTABLISH

```
   $ESTABLISH <file name>,SI
or $ESTABLISH <file name>,OI
or $ESTABLISH <file name>,BI,entry point
```

The effect of this command is to cause source, object or binary information to be read from the stream specified by the second parameter. In the case of a binary file, the third parameter is the octal address of the entry point of the program to be read in. The file name consists of a dollar sign or letter followed by any number of letters or decimal digits. Note that only the first six characters are significant. Source information is terminated by an end-of-file record or by the next command. In the case of the latter, the command is executed in the usual way.

Object information is terminated by an object end-of-file record. (Refer to Section 11, *External Record Formats.*) The following is given as an example:

```
$ESTABLISH TWO,SI
First record
Second record
$END OF TWO
```

This sequence of commands causes a new file, called TWO to be established with two records of source information in it.

The following limitation must be observed when establishing binary files:

(1) When executed, the file cannot be loaded into any location below '100.

(2) When executed, the file cannot be loaded into any location above:

```
for  8K machines -- '14777
for 12K machines -- '24777
for 16K machines -- '34777
for 24K machines -- '54777
for 32K machines -- '74777
```

## 3.15 EXECUTE

```
$EXECUTE <file name>,<list>
```

The effect of this command is to cause the named binary file to be loaded into core and the operating system to relinquish control by performing a JMP to the defined entry point.

The designated file must already contain a subsystem or user's program in absolute binary form. The operating system will input the list of parameters and store them in high core using three words per parameter. The number of parameters, excluding the file name, will be in location KPAR ('25). The address of the first parameter will be in location KADD ('26).

The maximum number of parameters allowed is given by $\left[\frac{\text{core size} - \text{B\$TC}}{3}\right]$ reduced to an integer. The parameters may be of two types, identifier or constant.

Parameters may be separated by a comma (,) or an equals sign (=) either of which may optionally be followed by one or more spaces. The parameter list is terminated by a single space after which all subsequent characters are ignored.

An identifier may consist of either an asterisk, a dollar sign or a letter followed by any number of letters or decimal digits. Only the first six characters are stored and these are stored at two characters per word. File names may not start with an asterisk. A constant consists of any number of octal digits (0 to 7). The first word of the parameter is zero and the other two words contain the least significant 32 bits of the constant.

For example, if the following command were input:

```
$EXECUTE  TRANSLATOR,*DICTIONARY=2,1377760
```

then the program in the binary user file TRANSL would be executed, KPAR would contain 3, KADD would contain B$TC+3 (say), and the parameter list would be stored in high core from B$TC upwards as shown in Table 3 - 3.

TABLE 3 - 3
EXAMPLE OF A PARAMETER LIST

| Location | Contents (Binary) | Remarks |
|---|---|---|
| B$TC | 11010100 11010010 | ISO Code for TR |
| B$TC+1 | 11000001 11001110 | ISO Code for AN |
| B$TC+2 | 11010011 11001100 | ISO Code for SL |
| B$TC+3 | 10101010 11000100 | ISO Code for *D |
| B$TC+4 | 11001001 11000011 | ISO Code for IC |
| B$TC+5 | 11010100 11001001 | ISO Code for TI |
| B$TC+6 | 00000000 00000000 | Constant |
| B$TC+7 | 00000000 00000000 | More sig. half |
| B$TC+8 | 00000000 00000010 | Less sig. half |
| B$TC+9 | 00000000 00000000 | Constant |
| B$TC+10 | 00000000 00000101 | More sig. half |
| B$TC+11 | 11111111 11110000 | Less sig. half |
| B$TC+12 | 00000000 00000001 | End of list |

The first word of each parameter is always negative for an identifier and zero for a constant. A positive word indicates the end of the list. If a subsystem is to return control on completion to the operating system, it may not overwrite any location between '22 to '77 or the top sector of memory. Interrupts will be enabled with all device masks reset on entry. Control may be returned to the operating system by the DAP-16 or FORTRAN statement CALL EXIT.

Should control be lost, for example, if a program goes into an infinite loop, the operator may abort the job manually. (See Section 6.)

## 3.16 FDISC

$FDISC <n>

This command means move forwards in the command file by the specified octal number of records as denoted by the octal number, n. Thus, the command pointer is incremented by the specified octal number, n. At the start of and during the execution of a command in a disc file, the command pointer always points to the next command in sequence. Therefore, the command $FD 0 has no effect and the command $FD 1 causes the next command to be skipped.

## 3.17 FIRST

$FIRST <file name>=<file name>,<. . .>

This command causes a file of the first name to be created which contains the first record of each of the other named files in sequence, ending with an 'end-of-file' record.

## 3.18 FORTRAN

$FN <source$_1$>,<source$_2$>,<listing>,<option>,<common>

This command causes the FORTRAN source program or programs in the source file <source$_1$>, to be translated into DAP-16 Mod 2 source. The DAP-16 Mod 2 source is held in the source file <source$_2$>, and the listing in the file <listing>. The <option> parameter is an octal constant, the value of which is set as described below. The <common> parameter is an octal constant specifying the base address for the FORTRAN 'COMMON' storage area (for further details of the use of this subsystem, see Series 16 FORTRAN Translator Manual).

Octal values of the option parameter may be 0, 2, 4, 6, 10, 12, 14 or 16 (0 default). If bit 13 of the parameter <option> is set, ie, the parameter has the value '00001X', the source output will be the complete translated FORTRAN source. If bit 13 is not set (ie options 0, 2, 4 or 6) the source output minimisation option of the input/output supervisor will be selected. The only records output to the source output file will be those essential to the operation of the translated program, ie assembler records. All comments (ie column 30 onwards) will be removed from these records, and all asterisk records will be suppressed. Listing output will not be affected.

This mode should not be used if assembler statements have been included in the program, which would go past column 28 (e.g., Macro calls or VFD pseudo-operations). These would otherwise be truncated to column 28, and no error indication would be given.

The least significant octal digit has the following effects:

| *Value* | *Effect* |
|---|---|
| 0 | TRACE will be incorporated only as specified in trace statements; optional statements will be ignored |
| 2 | TRACE will be incorporated only as specified in trace statements: optional statements will be compiled |
| 4 | All statements will be traced; optional statements will be ignored |
| 6 | All statements will be traced; optional statements will be compiled |

The default common base is the start of the BOS core resident routine.

It will be found much easier to use FORTRAN under BOS if a $DO job is written, tailored to the specific needs of the majority of users. This might, for example, contain the following steps, suitably coded into BOS commands starting with '%' characters:

1. *Input a FORTRAN source program on cards.*

2. *Translate the source program into DAP-16 code.*

3. *If there are any errors in the program, list the errors only, on the line printer, and abort.*

4. *Assemble the DAP-16 code.*

5. *If there are any DAP-16 errors, list the errors only, on the line printer, and abort.*

6. *Load the object code produced by 5.*

7. *If the system library does not contain all of the routines required to give 'loading complete' list the core map on the line printer, and abort.*

8. *Execute the FORTRAN program.*

## 3.19 IF

$IF <n>,<. . . >,<. . . >

The effect of this command is to cause the senselights specified in the parameter list to be tested. If all of the specified senselights are on (true) then the next command is obeyed, otherwise the next command is skipped. There are '30 senselights available to the user and these are numbered from right to left, from '1 to '30 respectively. (See also Section 3.37). There are also '30 virtual senselights available to the user and these are numbered from right to left from '41 to '70 respectively. A virtual senselight is always in the opposite state to its corresponding actual senselight.

For example, the command:

```
$IF 1,42,3
```

would be interpreted as: obey the next command if senselight 1 is set, senselight 2 is reset and senselight 3 is set, otherwise skip the next command.

## 3.20 JOB

$JOB <job name>

This command causes all user files remaining from the previous job to be deleted, and the user area of the backing store to be tidied up, that is, all the used blocks in the user library area are linked together in order. This action has no effect on the system files. System files are distinguished from user files by having a dollar sign ($) as their first character, for example, $SL, the system library file. Users may (subject to certain restrictions which may be placed upon them by those responsible for their installation), establish new system files, and manipulate them in exactly the same way as user files.

The job name is printed out on the console keyboard (ASR) and if specified on the line printer on a page by itself. The job name must be terminated by a space (or blank column), and could then be followed by the date, programmer's initials and account number or other similar information, for example:

```
$JOB TEST 2A 14. 1. 70 RLS 29806
```

## 3.21 KEY

$KEY <n>,<. . . >,<. . . >

The effect of this command is to cause the sense switches specified in the parameter list to be tested. If all of the specified sense switches are on (true), then the next command is obeyed, otherwise the next command is skipped. There are three sense switches numbered 2, 3 and 4 available to the user.

There are also three virtual switches available to the user and these are numbered 42, 43 and 44. A virtual sense switch is always in the opposite state to its corresponding actual sense switch.

For example, the command:

$KE 2,43,4

would be interpreted as: obey the next command if sense switch 2 is set, sense switch 3 is reset and sense switch 4 is set, otherwise skip the next command.

## 3.22 LOAD

$LOAD <list>

This command causes control to be transferred to the operating system relocating loader subsystem $LO, which scans the parameter list from left to right. The maximum number of parameters allowed is given by $\left[\frac{\text{core size} - \text{B\$TC}}{3}\right]$(see *Enter Operator (ASR) Mode, Section* 3.22.11). The loader parameters may be of three types:

(1) Loader commands

(2) File names

(3) Octal addresses

A loader command consists of an asterisk (*) followed by the name of the command. Loader commands may be written simply as an asterisk followed by the first letter of the command, for example:

*ADDRESS

may be written as

*A

File names may not start with an asterisk (*). If the first character of the file name referred to is a dollar sign ($), the file is a system file; if the first character is a letter, it is a user file.

The object code being loaded must not overwrite any core locations below '100 (see Section 3.22.20, *Memory Overflow*). The possible loader instructions are detailed in Sections 3.22.1 to 3.22.20.

### 3.22.1 Set Initial Address

This consists of two parameters, of the form:

*ADDRESS=<octal address>
or
*A=<octal address>

If this instruction is used, no object files must have been loaded since the last time the loader was initialised.

If no *ADDRESS instruction is given, an initial address of '1000 will be assumed. Relocatable programs will be stored starting at this address.

To force an initial address of zero, *A=100000 must be input.

## 3.22.2 Set Base Address

This consists of two parameters, of the form:

*BASE=<octal address>  or  *B=<octal address>

This instruction may be given at any point in the load operation. If no *BASE instruction is given, a base address of '100 will be assumed. Cross-sector references will be stored starting at this address.

## 3.22.3 Set COMMON Base Address

This consists of two parameters, of the form:

*COMMON=<octal address>  or  *C=<octal address>

If this instruction is used, it must occur before the first instruction to load an object file, regardless of whether or not the loader has been re-initialised.

If no *COMMON instruction is given a COMMON base address equal to the address of the BOS Core Resident routine will be assumed. COMMON storage will be allocated downwards from, but not including, this location.

If a load into virtual memory (see Section 3.22.17) is being performed, and the COMMON base address is to be raised above its default value, the *COMMON instruction must precede the *VIRTUAL instruction to ensure that sufficient virtual memory scratch area is allocated to contain the code being loaded.

When loading a segment of a segmented FORTRAN program, the COMMON base address must be set below the location B$FP. This is normally located at 'X5000 (where X = 2, 3, 5 or 7 depending upon core size – see Section 5.2.2 'Loading the BOS Main System') thus, the COMMON base address may be set at 'X4777. This address should be set below B$EF (normally located at 'X6000) for systems using 95 word blocks.

## 3.22.4 Debug

This instruction consists of:

*DEBUG  or  *D

This instruction causes the binary file $DE to be loaded into core, overwriting part of the loader subsystem. $DE contains a debugging package which may, for example, consist of DEBUG, X16DEBUG, or any other suitable program. The entry address of the debug package will be typed out, followed by a re-entry address for the loader subsystem, then the debug package will be entered. For further details of debugging commands, see the listings of the appropriate programs. The prime purpose of re-entry to the loader subsystem is to enable a patched result file to be dumped. When the loader is re-entered after obeying an *DEBUG command, it will obey the rest of its line of parameters and go automatically into operator mode (see Section 3.22.11). Accordingly, the only commands accepted by the loader after re-entry are *DEBUG, *EXECUTE, *OPERATOR, *RESULT and *QUIT (see Section 3.22.13), as the debug package will usually overwrite most of the loader routines. The *DEBUG command is not valid while the loader is operating in Virtual Memory Load (see Section 3.22.17).

### 3.22.5 Execute

This command may be written as:

*EXECUTE or *E

If loading is complete, the program will be executed in the addressing mode in which it was loaded, starting at the *START address from the core map (see *Set Entry Address*, Section 3.22.15).

If loading is not yet complete, a system error LO will be generated and the job will be aborted.

If there are any parameters following the *EXECUTE command, they may be accessed by the object program through the BOS low-core common locations KPAR and KADD, as if they had been parameters following a BOS $EXECUTE command.

The *EXECUTE command is not valid whilst the loader is operating in Virtual Memory Load Mode (see Section 3.22.17). The EXECUTE command must not be used to execute any segment of a segmented FORTRAN program (see Sections 7.1.3 and 7.1.4). A result file must be stored using the RESULT command (see Section 3.22.14) and subsequently executed using the BOS $EXECUTE command.

### 3.22.6 Force Load

This instruction consists of one parameter, and may be written:

*FORCE or *F

Where this instruction occurs, there must have been at least one instruction to load an object file since the loader was last initialised. Its effect is to ensure that the next object module will be loaded, even if it has not yet been called.

### 3.22.7 Re-initialise Loader

This instruction consists of one parameter, and may be written as:

*INITIALISE or *I

Its effect is to set the loader to its initial state. The names of any subroutines already loaded will be deleted from the loader symbol table, all *BASE entries and the *BOTM, *LOW, *START, *HIGH and *TOP addresses will be cleared out, the initial address and base address will be set to their initial default values, and the *COMN address will be set to its previous initial value, whether this be a value specified by a *COMMON instruction or the default value. Any code already loaded into memory will, however, remain there.

After an *INITIALISE instruction, an *ADDRESS instruction may be used to set a new initial address before the next file is loaded, but a new *COMMON instruction may not be given.

It is not necessary to give an *INITIALISE instruction at the start of every $LOAD operation, as the first initialisation is automatic.

### 3.22.8 Enter Library Load Mode

This instruction consists of one parameter, and may be written as:

```
*LIBRARY
   or
*L
```

It may be given at any point during the load operation. In library load mode, which is the normal mode of operation of the loader, the first object module processed by the loader after initialisation will always be loaded; thereafter, object modules will be loaded only if called by a module previously loaded, or if specifically force-loaded by use of the *FORCE command. The loader subsystem is initially in library load mode (see *Enter Total-Load Mode*, Section 3.22.16).

### 3.22.9 Core Map

This instruction consists of two entries in the parameter list, and takes the general form:

```
*MAP =<file name>
     or
*M =< file name>
```

A core map is generated and stored in the named file, which must not previously have existed. More than one core map may be generated during the loading process, but they must all be in different files.

The core map is similar to that generated by the off-line loader LDR-APM but the following special points about the core map produced by the BOS loader should be noted:

(1) *BOTM gives the lowest location loaded.

(2) *LOW gives the lowest location occupied by the coding of any program loaded.

(3) *HIGH gives the first free location above any program coding loaded.

(4) *TOP gives the first free address above any location loaded.

NOTE:

The word 'loaded' above means actually set to a value during the load process.

Locations loaded include program instructions, desectoring links, program constants, arrays set by a BSZ pseudo-op, and BLOCK DATA. Arrays set by a BSS pseudo-op, and ORG pseudo-ops, have in themselves no effect on *BOTM and *TOP, but will affect *HIGH and *LOW.

The words 'program coding' above include program instructions, program constants, arrays set by a BSS or a BSZ pseudo-op, and the locations specified in ORG pseudo-ops. Desectoring links and BLOCK DATA are not included.

*LOW and *HIGH have thus exactly the same meaning as they have in the core map produced by LDR-APM rev. D.

### 3.22.10 Load in Normal Desectoring Mode

This instruction consists of one parameter, and may be written as:

```
*NORMAL
   or
*N
```

This instruction may occur at any time during the load operation. Its effect is to cause the object programs being loaded to be desectored in the non-extended-addressing (LXD mode), unless otherwise specified by an enter-extended-desectoring-mode object block subsequently encountered.

If no *NORMAL instruction (or *XTENDED instruction, see Section 3.22.18) is given, the loader will desector in the mode corresponding to the addressing mode in which the governing BOS system operates.

When the loader is re-initialised, the desectoring mode will revert to the mode last specified by a *NORMAL or *XTENDED instruction or, if no such instruction exists, to the default mode.

### 3.22.11 Enter Operator (ASR) Mode

This instruction consists of one parameter, written as:

```
*OPERATOR
    or
*O
```

It may be given at any point during the load operation. Its effect is to cause the loader, when it has finished scanning the current line of parameters, to request input of further parameters from the ASR instead of returning control to BOS.

The loader requests parameter input by outputting a double question mark to the ASR. Parameters should be typed in exactly as they would be for the original $LOAD command -- but the new line of parameters must not commence with $LOAD. Up to 32 parameters may be typed on the new line, which should be terminated by typing a carriage return character. Erroneous characters just typed will be successively deleted by typing a series of left arrow (←, shift O) characters, and a whole line may be cancelled by typing a commercial at (@, shift P) character. Other non-printable characters are ignored. The only legal printable characters are letters (A-Z), digits (0-9), space, dollar ($), asterisk (*), comma (,) and equals (=); other printable characters generate a system error LO and the job will be aborted if they are not cancelled before carriage return is typed.

Once the loader is in operator mode, it will continue to return to the ASR for more parameters at the end of every line, and will not return control to the BOS system until a *QUIT command, or an error, is encountered.

### 3.22.12 Program Break

This instruction consists of two entries in the parameter list, of the form:

```
*PBRK=<octal address>
        or
*P=<octal address>
```

When this instruction is used, at least one instruction to load an object file must have occurred since the last time the loader was initialised. The purpose of the *PBRK instruction is to cause the next relocatable program to be stored starting at this address -- for example, to reduce the number of cross-sector indirect references by starting the loading of another file at the beginning of the next sector.

## 3.22.13 Quit

This instruction consists of a single parameter, written as:

```
*QUIT
  or
*Q
```

It may occur at any time. When it is obeyed, the BOS Main System is recalled and control is returned to BOS. The *QUIT command should be used to terminate use of the loader sub-system while in operator mode.

## 3.22.14 Store Resulting Binary File

This instruction consists of two or more entries in the parameter list, and takes the form:

*RESULT=<file name>,<$a_1$>,<$b_1$>,<$a_2$>,<$b_2$>, . . . .<$a_n$>,<$b_n$>,<s>
(where a, b and s are octal addresses)
or
*R= etc.

A new file is created on the backing store, with the given file name. This file will contain a core image of locations $a_1$ to $b_1$ inclusive, locations $a_2$ to $b_2$ inclusive, . . . . and locations $a_n$ to $b_n$ inclusive. The parameter(s) will be stored in the file as the entry address of the program, to enable it to be run by a BOS $EXECUTE command. It will not alter the *START address stored for the core map.

If any of the parameters'a' iszero, it will be replaced by the current *BOTM address from the core map. If any of the parameters 'b' is zero, it will be replaced by an address one short of the current *TOP address from the core map. If n=0, i.e., there are fewer than two octal parameters, a core image of locations *BOTM to *TOP-1 will be copied to the result file.

If the parameter 's' is not present, i.e., there is an even number of parameters, then if, and only if, loading is complete (there are no outstanding external references) the *START address from the core-map will be substituted for 's'. However, if 's' is absent and loading is not complete, no entry address will be filed, and the result file will not be executable by a BOS $EXECUTE command, although it may still be output via the binary output stream.

The result file parameter list is considered terminated when the first non-octal parameter is found, or (except in operator mode) when the end of the line of parameters is reached.

If the *RESULT command is used, at least one instruction to load an object file must have occurred since the last time the loader was initialised.

In Virtual Memory Mode (see Section 3.22.17) it is possible to load into any location from 0 to '77776 inclusive, and a result file may be generated consisting of a core image of any of these locations. However, the result file may not always be executable by means of a BOS $EXECUTE command. If the result file contains a dump of core locations below '100 or above the address of the BOS Main System file pointer table, system failure will be caused if execution under BOS is attempted.

A result file may, however, be output via the binary output stream and executed off-line.

### 3.22.15 Set Entry Address

This instruction consists of two parameters, of the form:

```
*START=<octal address>
          or
*S=<octal address>
```

It may occur at any point during the load operation. The octal address specified will then be the address at which the program loaded is entered in response to an *EXECUTE command, and will appear on the core map as the *START address. It will be output to a binary result file if loading is complete, unless overridden by the *RESULT command entry address parameter.

If no *START address is specified, the start address of the program loaded will be the address specified in the END object block of the first module loaded after loader initialisation or, if no such address is specified, the *LOW address of that module.

### 3.22.16 Enter Total-Load Mode

This instruction consists of a single instruction, written:

```
*TOTAL
   or
*T
```

It may be given at any point during the load operation. In total-load mode, every object module processed by the loader will be loaded, regardless of whether or not it has been called (see *Enter Library Load Mode*, Section 3.22.8).

### 3.22.17 Virtual Memory Load

When loading into core, there are limitations imposed on the range of load operations which may be performed. For example, a program may not be loaded into locations below '100, as this would corrupt BOS low-core common; neither may a program be loaded into locations occupied by the loader subsystem (see *Memory Overflow*, Section 3.22.20). These limitations may be overcome by performing a load into virtual memory. If the instruction:

```
*VIRTUAL
    or
*V
```

is given, a file VTMEM$ will be established in the user area of disc, after first checking that no such file exists as a result of a previous manually interrupted run, and deleting it if it does. (The file name VTMEM$ is, by reason of the dollar sign in column 6, illegal if input to BOS by an operator). The file VTMEM$ contains the equivalent of up to 32,768 words of memory, initially zero, and all subsequent object files will be loaded into the virtual memory file, any location of which may be addressed. The program being loaded cannot be executed while in virtual memory, and neither can the *DEBUG command be used; but a core map may be obtained, and a result file may be generated in order to produce a self-loading system tape via the binary output stream. The result file may also be executed by means of a BOS $EXECUTE command, subject to the limitations of addressing mode, and the fact that it must not overwrite certain parts of the BOS system while being read into core. The file VTMEM$ will be deleted when the loader returns control to the BOS system.

The size of virtual memory allocated is such that the highest location available has an address equal to the value of the common base at the time when the *VIRTUAL instruction was given. The default value is the address of the BOS core-resident routine. Thus, a *COMMON instruction *raising* the value of the common base address must be given before, and not after, the *VIRTUAL instruction. If this condition is not complied with, and the loader attempts to load into a non-existent virtual memory location, a system error MO will be generated, with no other diagnostic, and the load will be aborted immediately. In contrast with other memory overflow conditions, no memory map may be generated after this error has occurred.

The *VIRTUAL instruction, if used, must occur before the first instruction to load an object file, regardless of whether or not the loader has been re-initialised.

By using a *COMMON instruction to re-adjust the COMMON base to any convenient location, up to and including '77777, a virtual memory load may be used to load a program which is actually too large to fit into the available core on the computer being used to load it.

## 3.22.18 Load in Extended Desectoring Mode

This instruction consists of one parameter, and may be written as:

```
*XTENDED
    or
*X
```

This instruction may occur at any time during the load operation. Its effect is to cause the object program being loaded to be desectored in the extended addressing (EXD) mode, unless otherwise specified by a leave-extended-desectoring-mode object block subsequently encountered. If no *XTENDED instruction (or *NORMAL instruction, see Section 3.22.10) is given, the loader will desector in the mode corresponding to the addressing mode in which the governing BOS system operates.

When the loader is re-initialised, the desectoring mode will revert to the mode last specified by a *NORMAL or *XTENDED instruction or, if no such instruction exists, to the default mode.

## 3.22.19 Load an Object File

The instruction in this case consists simply of one parameter, the file name. Any object programs in the file that have previously been called are loaded. Others are merely ignored by the loader, after checking for validity (but see *FORCE, *LIBRARY and *TOTAL, Sections 3.22.6, 3.22.8 and 3.22.16 respectively).

Before loading the first object file into core memory (i.e., if not operating in virtual memory mode), the loader clears core from location '100 up to the bottom of the loader. The locations occupied by the virtual memory handling routines – just over five sectors – will be freed and cleared for normal core loading purposes.

## 3.22.20 Memory Overflow

When loading into core memory, the loader detects memory overflow if one of the following conditions exists:

(1) An attempt has been made to overwrite one of the locations 0-'77, which are reserved for BOS low core common. The *BOTM address will be less than '100.

(2) An attempt has been made to overwrite the loader symbol table, or any location above the bottom of the loader. The *TOP address will be greater than the *NAMES address.

(3) Code or desectoring links have overlapped COMMON storage. The *HIGH address or the *BASE address will be greater than the *COMN address.

(4) A table of desectoring links has overflowed a sector boundary. The offending *BASE address will be 'XXOOO.

(5) A string created in memory by the loader has been overwritten. If the string contains unsatisfied subroutine calls, the error will normally be detected when the subroutine in question is loaded and defined. The error will be notified by the message SØ, which will be typed on the ASR instead of MO (see below) and, in addition, will appear on the core map.

If the memory overflow condition occurred whilst the loader was generating a desectoring link, the error will be notified by the message BO, which will be typed on the ASR and, in addition, will appear on the core map. When loading in virtual memory mode, the virtual memory routine detects an error if an attempt is made to load into a non-existent virtual memory location. This error can only be due to a *COMMON instruction raising the common base address occurring *after* the *VIRTUAL instruction. If this particular error occurs the load will be aborted immediately. The only output seen on the ASR will be:

```
SE MO
```

as opposed to the usual message of the form:

```
MO
SE MO
```

and it will not be possible to obtain a core map after the error has occurred.

If BLOCK DATA being loaded into COMMON has attempted to overwrite the loader, the condition may be remedied by re-arranging the order in which COMMON blocks are stored, by lowering the COMMON base with an *COMMON instruction, or by performing a virtual memory load. Conditions (1) and (2) above may also be alleviated by performing a virtual memory load. When loading into virtual memory, the loader detects memory overflow if condition (3), (4) or (5) above exists or, additionally, if the loader symbol table attempts to extend below location '100 (in core memory).

When memory overflow is detected, the loader types 'MO', 'BO' or 'SO' on the ASR and passes into memory overflow abort mode. In this mode, all instructions to load an object file are skipped. When an *EXECUTE, *INITIALISE, *RESULT, or *QUIT command is encountered, when the end of the parameter line is reached (unless in operator mode), or when any *other* error is detected, the load operation will be aborted and control returned to BOS, giving a system error MO. In this way, it is still possible to file a core map after memory overflow has occurred.

The user should note that in some circumstances the core map will show the same addresses after a memory overflow has occurred as it was showing just before the error. For example, the *BASE entries show the *first free* location in a base sector. If sector 15 is completely full with desectoring links, the core map entry for that sector will be:

```
*BASE      16000
```

This condition is not an error. If, however, an attempt is made to store another desectoring link in sector 15, a memory overflow will be detected, but the entry for sector 15 will still read as above.

The loader cannot in general detect code or desectoring links overwriting other code or desectoring links.

### 3.22.21 Go to Next Sector

This instruction consists of one parameter, and may be written as:

*GTNS or *G

When this instruction is used, at least one instruction to load an object file must have occurred since the last time the loader was initialised. The purpose of the *G instruction is to cause the next relocatable program to be stored starting at the beginning of the next sector. If the loader is already positioned at the start of a sector the *G instruction has no effect.

### 3.22.22 Enter Work File Mode

```
*WORKFILE = <file name>
or
*W = <file name>
```

It may be given at any point during the load operation. Its effect is to cause the loader to suspend the input of parameters from the current parameter list and to input further parameters from the file specified. The work-file may consist of any number of records. Each record can consist of a maximum of 32 loader parameters. When the work-file parameters have been exhausted the loader returns to the point in the original parameter list following the initial work-file actioned. If a work-file has as a parameter a *WORKFILE instruction the loader will leave the current work file and input from the new file specified. When the End of File is reached the loader will not return to the previous work-file but will return to the original parameter list before the first *WORKFILE instruction was obeyed.

Some examples of Loads using work-files:

```
(1)  $LO *W = W1

(2)  $LO *V, *W = W1, *M = M1, *R = R1

(3)  $LO *V, *W = W1, *O
     ??
     *W = W2, *Q

(4)  $LO *W = W1, *W = W2, *W = W3
```

### 3.22.23 Undefined Map Symbols

A map of undefined symbols can be obtained by outputting the map file to the BOS error output stream. ie $OUTPUT MAP, EO.

## 3.23 MAKE

$MAKE <file name>=<list>

This command causes the source or object information in one or more files named in the list to be joined together to make a new file. The old files are not destroyed. For example:

$MAKE X=Y,Z

causes a new file (X) to be made by joining together the contents of files Y and Z, in that order. Again:

$MAKE MYLIB=O1,O2,O3,$SL

causes a user library, MYLIB, to be made up from the object files O1, O2 and O3 and the system library $SL.

## 3.24 MBACKWARD

```
$MBACKWARD <device name>,<no of files>
```

This command is used in connection with magnetic tape input and output. The device name should be M0, M1, M2 or M3. The particular magnetic tape transport is made to move the magnetic tape backwards until the stated number of files has been passed. The second parameter is an octal number.

For example, to read again a file which has just been ESTABLISHed, from magnetic tape unit no. 2, the following commands are necessary:

```
$MBACKWARD M2,1

$ESTABLISH FV,SI
```

If the number of files is zero then the magnetic tape device is positioned at the start of the current file. If the number of files is negative (e.g., '177776), then the magnetic tape device is backspaced by the specified number of files (e.g., '177776).

## 3.25 MFORWARD

```
$MFORWARD <device name>,<no of tape marks>
```

This command is the converse of MBACKWARD, the previous command. The magnetic tape on the specified transport is moved forward until the specified number of tape file marks has been passed. The last tape mark passed will not be read if an ESTABLISH command is given next.

## 3.26 MREWIND

```
$MREWIND <device name>
```

This command causes the magnetic tape on the specified transport to be rewound. The device name should be M0, M1, M2 or M3. The operating system inputs and executes the next command without waiting for the rewind to be completed. If required, all four transports may be commanded to rewind simultaneously by executing four MREWIND commands. If other commands, such as OUTPUT or MFORWARD are executed for transports that are rewinding, the correct delay will be inserted before implementing the command. This command is obeyed even when the system error marker is set. Note that this command causes the file position indicator to be reset to 1 (see *POSITION Command*, Section 3.34).

## 3.27 NAMECHANGE

```
$NAMECHANGE <file name 1>=<file name 2>
```

The effect of this command is to cause the name of the file specified by file name 1 to be changed to that specified by the <file name 2>. The first named file must already exist and the new file name must not be identical to the name of any existing file. Both the files named in this command must be either system or user files, i.e., it is illegal to attempt to change the name of a system file to a user file or vice versa.

## 3.28 NEXT

$NEXT

The effect is to copy one record of source information from the command input stream to the source output stream. For example, the commands:

```
$NEXT

$WAIT
```

would cause the $WAIT record to be copied from the second card to the SO (source output) stream; the $WAIT command would not be executed at this point, but could be executed later, if the SO output were read in again on the CI (command input) stream.

## 3.29 NGROUP

$NGROUP <file name 1>=<file name 2>,<. . . .>

This command is used for removing 'end-of-group' records from existing file(s). Execution of this command causes a file of the first name to be generated which is a record by record copy of the subsequent named file(s). The new file excludes any 'end-of-group' records which may have existed in the previous files.

## 3.30 NTRACE

$NTRACE

This command turns off command tracing (see *$TRACE Command,* Section 3.42).

## 3.31 NOTEMPTY

$NOTEMPTY <file name>

The effect of this command is to cause the first record of the named file to be examined. If it is an 'end-of-file' record, the next record input on the command input stream is skipped. If it is not an 'end-of-file' record, the next record input on the command input stream is obeyed (if it is a valid command). All 'end-of-group' records are ignored. If the file does not exist then it is deemed to be empty and the next command is skipped.

## 3.32 OUTPUT

$OUTPUT <file name>,<stream name>,<optional bootstrap address>

This command causes the contents of the named file to be output on the named stream, which must be either EO (error output), SO (source or listing output), OO (object output) or BO (binary output).

When EO is specified, only error records will be output. For example, error lines from a DAP-16 assembly. When SO is specified, only source and error records will be output. On the OO stream, only object records will be output, and on the BO stream, only binary records will be output. Binary records must not appear anywhere in files output on the EO, SO or OO streams.

For Binary Output the optional parameter may be used to indicate the *sector* in which the bootstrap will reside. If this parameter is not included, the bootstrap will be located as close as possible to, and above, the area of core to be occupied by the binary file being output.

Output to the line printer will be divided into pages of 37 lines each, suitable for printing on fanfold stationery 8½ inches deep. (This figure can be changed, if desired, when setting up a configuration but not between jobs.) Each page will have as its heading the first line in the current file, and each group will start a new page. Pages will be numbered from one within each file.

Output to other source devices (e.g., the paper tape punch, PP) will be unbroken, in a form suitable for subsequent re-input.

## 3.33 OutputLINENUMBERS

$OLINENUMBERS <file name>,SO

This command is identical to $OUTPUT except that line numbers will be added to each record output. Line numbers start at 1 and are incremented by 1 for each record output. The only meaningful output streams for this command are Source or Error (see *OUTPUT Command*, Section 3.32).

## 3.34 POSITION

$POSITION < mag tape unit n>

The effect of this command is to cause magnetic tape unit 'n' to be positioned at the start of the file specified by the next record to be read on the source input stream.

After the magnetic tape has been repositioned, the current file position indicator is amended so that it points to the file following the request file. Note that the command, MREWIND ($MR) causes the file position indicator to be set to one. There are two legal formats for the input stream record and these are:

| | |
|---|---|
| XX | Position mag. tape to the start of file XX |
| XX,YY | Position mag. tape to the start of file XX |

The next $POSITION command obeyed causes the file position indicator to be incremented by one without reading an input record. This sequence is repeated until the file position indicator exceeds YY.

If XX is zero, the magnetic tape unit is rewound and the next record input on the command input stream is skipped.

NOTES:

(1) XX and YY are DECIMAL numbers.

(2) The commands $MBACKWARD, $MFORWARD do not alter the file position indicator used by $POSITION commands. These commands should not be used at the same time as $POSITION commands or the file position indicator will not agree with the actual magnetic tape file position.

(3) The $POSITION command assumes a file is to be established from the magnetic tape following the command. The use of multiple $POSITION commands without any $ESTABLISH command will result in the tape being mis-positioned.

(4) The $POSITION command file position indicator will assume that all $POSITION and $MREWIND commands are addressed to the same magnetic tape unit. The $POSITION command should not be used to position multiple units if later re-positioning of the same unit is required.

(5) The $POSITION command should be preceded by a $MREWIND command to initialise the file position indicator. (This is only required before the first $PO command.)

## 3.35 RESET

$RESET

This command has no effect under normal circumstances. However, if a system error has been detected by the operating system, the system error marker will be reset, so that subsequent commands will be executed. When the system error marker is set, only $ATTACH, $JOB, $REWIND, $DO, $NEXT, $RESET, $TYPE and $WAIT commands will be executed. $NEXT and $TYPE commands cause the next command input record to be ignored; other commands have their usual effects.

Note: This does not apply if command input is attached to the ASR keyboard when a system error is detected. In this case, the system error marker will be automatically reset and the $RESET command need not be input.

## 3.36 RUNOUT

$RUNOUT

This command causes a length of leader (about 20 inches) to be punched on the high-speed paper tape punch, and then turns the punch power off.

## 3.37 SENSELIGHTS

$SENSELIGHTS <octal parameter 1>,<octal parameter 2>

The effect of this command is to cause the dedicated senselight word to be extracted from core and a logical AND operation to be performed on it using octal parameter 1. An exclusive OR operation is then performed on the result using octal parameter 2. Note that both octal parameters must be in the range: '0-'77777777.

## 3.38 SFILE

$SFILE <old master>,<new master>

In operation, this command is similar to $SHORTEN except that records are not truncated to a maximum of 72 characters. Source or Error Records from the old master are scanned from the right hand end to see if the trailing words contain spaces. Trailing spaces are removed from these records and the record length adjusted accordingly. These corrected records are then output to the new master.

## 3.39 SHORTEN

```
$SHORTEN <old master>,<new master>
```

This command is used to shorten source records to a maximum of 72 characters, omitting material which cannot be printed on an ASR printer. The old master file, named first in the command, must already exist and the new master file must not yet exist. The old master file may not contain binary records; if any are found, a system error occurs. Any end-of-group marks, object or special records are copied unchanged from the old master to the new master. Any source or error records are examined; if they contain more than 72 characters, they are reduced to 72 by omitting characters on the right; the remaining characters are then copied into the new master in order.

This command may be used to reduce considerably the size of a file in which each record contains card identification numbers in columns 73 to 80.

## 3.40 SINGLE

```
$SINGLE <file name>
```

The effect of this command is to cause a single record to be read from the source input stream and the named file to be created which consists of the single input record and a file mark.

## 3.41 TIDY

$$\text{\$TIDY} \begin{bmatrix} \text{U} \\ \text{S} \end{bmatrix}$$

This command causes all unused blocks in the file area specified to be re-linked into a continuous free block area. The existing files are checked for integrity and, if all is well, are left undisturbed. If any inconsistency is found the file is deleted and the space occupied by the file is returned to free storage. A message is typed to indicate that the file has been deleted. If $TIDY finds a block which consistently gives hardware errors when attempting to read it, it is deemed to be faulty. Such faulty blocks are not linked into the free block chain, and become inaccessible by the system. Should such a block be found in a user file, the file is deleted, and the remaining blocks are returned to free storage. A message is typed to indicate that the file has been deleted. The optional parameter 'U' or 'S' may be given to restrict the $TIDY to the 'User' or 'System' file area respectively. If neither 'U' or 'S' is specified, a TIDY of both User and System file areas will be actioned. A $TIDY command should be executed whenever disc overflow occurs (DO error) or whenever mixed file errors occur (MF).

## 3.42 TRACE

```
$TRACE
```

This command initiates command tracing. In this mode, every command which BOS actually attempts to execute is typed out on the ASR. (This does not apply to commands input from the ASR but does apply to commands input from a disc file.) This includes commands constructed by a $COMMAND command.

For example, if in command trace mode and the following command is executed:

```
$CO F1, F2, F3
```

and files F1, F2, F3 contain as their first records ES, X, SI respectively, the following would be output on the ASR:

```
** $CO  F1,F2,F3

** $ES  X,SI
```

Commands that are skipped under any circumstances (e.g., if the ABORT flag is set) are not printed out.

## 3.43 TYPE

$TYPE

The effect of this command, which has no parameters, is to copy the next command input record and print it on the ASR. Even if the record contains a dollar sign in column 1, it will not be executed.

For example:

```
$TYPE

LOAD LP WITH 3-PART STATIONERY PLEASE

$WAIT
```

This causes the message to the operator to be typed out on the ASR. The operating system then waits for the operator to press the RETURN key CR ('015 or '215) before reading the next command input record.

## 3.44 EDIT

$EDIT <om>, <nm>, <um or '1'>, <pm or '1'>

This command calls in the Edit subsystem, $ED, to edit an existing file or old master (om) to produce a new file called the new master (nm). The edit instructions can be either contained in an existing file called the update master (um), or input via the ASR teletype keyboard, indicated by a '1' in the $ED call. If the edit PRINT command (see EDIT manual) is to be used, the print output can be either to a file, called the print master (pm), or on the ASR teletype, indicated by a '1' in the $ED call.

The update stream and the print stream are both optional parameters. The default device for both streams is the ASR.

A brief summary of Edit commands is enclosed in Appendix D, Table D-3.

## 3.45 VERIFY

```
$VERIFY <filename>,<file name>
```

This command enables the user to verify whether two files on the disc contain identical information.

The two files may, for example, have been created by ESTABLISHing the first file, TYPEing a message to the operator, WAITing for him to reposition the input and then ESTABLISHing the second file. The VERIFY command then allows the user to check that the two files are identical; if they are, then the probability of a misread is virtually negligible. This method of working is recommended for use whenever valuable data is to be read from or written to any device. This is especially applicable to paper tape and card devices.

If the two named files contain absolutely identical information, the operating system goes on to read the next command. However, if there is any discrepancy whatsoever, a system error will occur. The nature of the discrepancy can be determined in general terms from the exact error message given (RF,VF, VG, VT, VL, VA or VE). Refer to Appendix A for full details of ERROR MESSAGES.

All types of records may be verified and in any mixture, including binary files, whether formed by loading or by ESTABLISHing a file from a self-loading system tape. In order to verify source files on magnetic tape, the files must be no-grouped.

## 3.46 WAIT

```
$WAIT
```

This command, which is obeyed whether or not the system error marker is set, causes one record to be input from the console keyboard (ASR) irrespective of which device is currently assigned to the command input stream. If the type-in has a dollar sign in the first column, it will be executed as a command in the usual way.

The WAIT command may be used by the programmer and the computer operator in several different ways:

(1) The programmer causes a message to be typed out, using the TYPE command, and waits until the operator has taken appropriate action; the operator then presses the RETURN key (CR, '015 or '215).

(2) In instances where the command input stream is assigned to the paper tape reader, the WAIT command should appear at the end of all paper tape in order to allow the operator to change tapes.

(3) In the case of (2), the operator may type in: $AT CI,AK in order to retain permanent control from the console keyboard (ASR).

(4) The operator may type in $AB (Abort) and, if so desired, start the next job.

## 3.47 MACRO

```
$MC <file 1> = <file 2>, <file 3>, . . . . <file n>
```

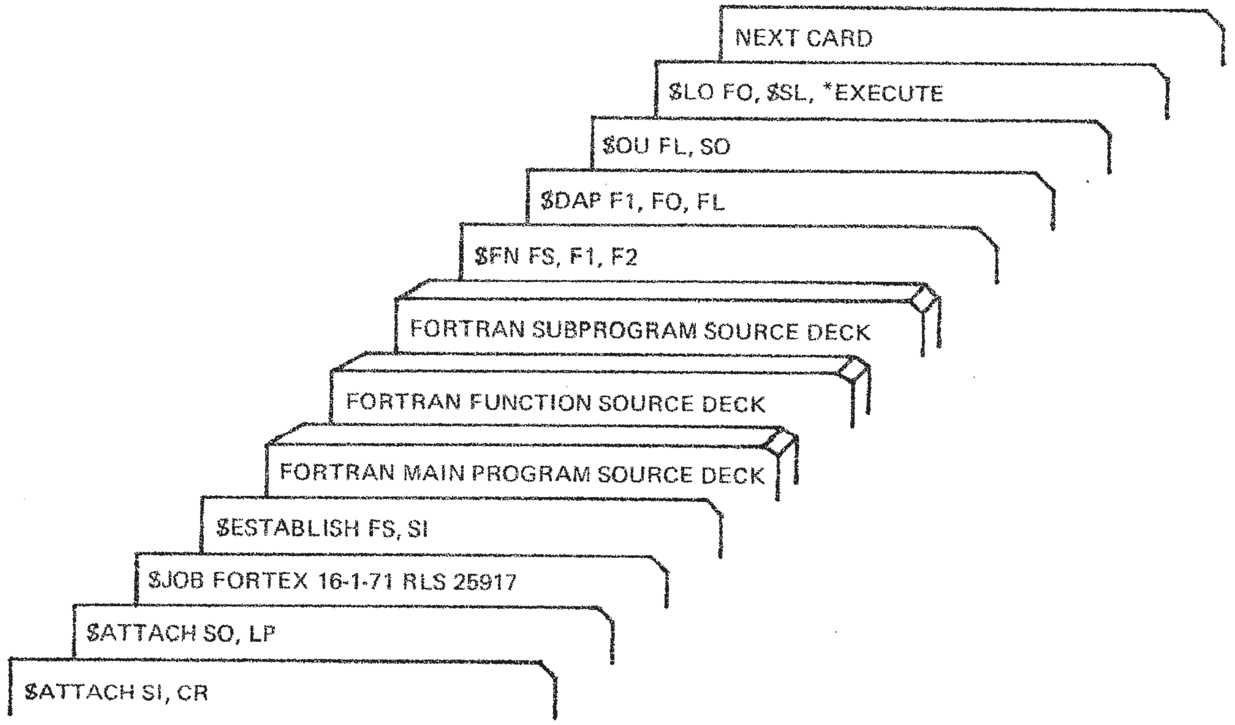
This command causes the DAP-16 files named file 2, file 3, etc., to be processed by the macro pre-processor to produce a new source file, file 1, which contains the expanded source. The first files named in the list, file 2, file 3, etc. may contain libraries of frequently used macros.

# 4 EXAMPLES OF COMPLETE JOBS

The following examples illustrate how source programs and control commands (cards) may be combined to perform various complete jobs. For the purposes of illustration, the examples depicted here are in the form of punched cards, but could equally well be read in from punched-paper tape or magnetic tape.

## 4.1 FORTRAN Compilation and Execution

In this example, source input has been assigned to be from the card reader and source output (listing) to the line printer. The job name is FORTEX, the other information on the job card merely being printed out on the line printer. Fig. 4 - 1 shows the card deck for this job.


NEXT CARD
$LO FO, $SL, *EXECUTE
$OU FL, SO
$DAP F1, FO, FL
$FN FS, F1, F2
FORTRAN SUBPROGRAM SOURCE DECK
FORTRAN FUNCTION SOURCE DECK
FORTRAN MAIN PROGRAM SOURCE DECK
$ESTABLISH FS, SI
$JOB FORTEX 16-1-71 RLS 25917
$ATTACH SO, LP
$ATTACH SI, CR

CARD DECK FOR JOB FORTEX FIG. 4 - 1

The FORTRAN main program, function and subprogram source decks are all stored in a file FS which is then translated to give a DAP-16 source file, F1, and a listing file F2. The DAP-16 source file is then assembled to produce the object file FO, and the listing file FL. At this point, the assembled program listing, FL, is output on the line printer. The object file FO is then loaded followed by the System Library, $SL. The program is then run, starting at the first executable instruction in the main program. When the program is finished, the FORTRAN instruction CALL EXIT is obeyed and the operating system will continue as directed by the next control command (card).

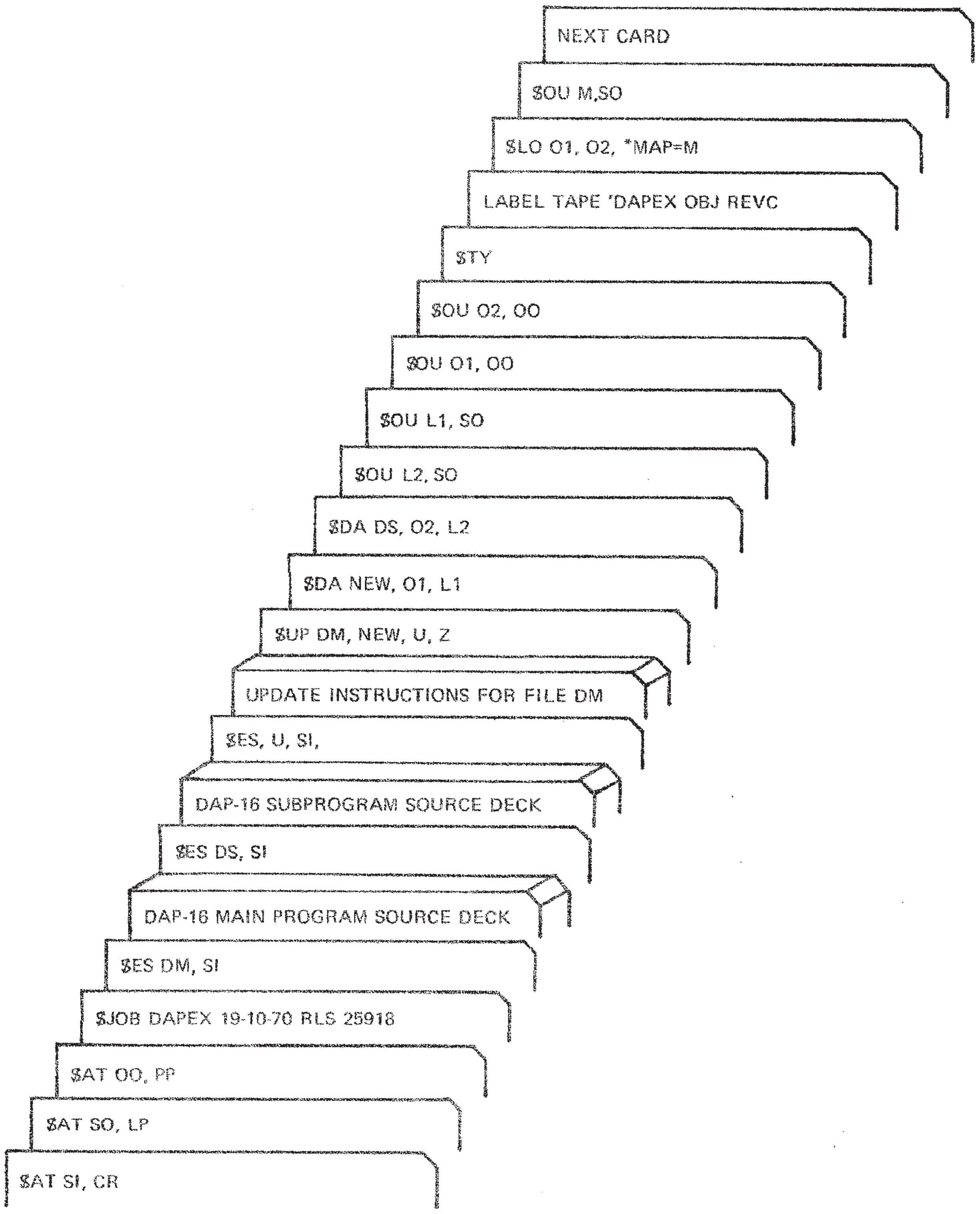
## 4.2 DAP-16 Assembly

In this example, source input has again been assigned to be from the card reader, source output (listing) to the line printer, and object output to the paper-tape punch. The job name is DAPEX, the other details being only comments. Only the first two characters of a control command are necessary and subsequent characters are usually omitted, as shown in the following example. Fig. 4 - 2 shows the card deck for this job.

The DAP main program is stored in a file called DM and the subprogram source decks in a file called DS. Since DM is known to contain errors, it is updated to form a new file called NEW.

The next two cards each cause a DAP-16 assembly to be executed, thereby producing separate files, O1 and O2, and listing files L1 and L2. The latter files are subsequently listed on the line printer and the object files are punched on the object output (OO) device (paper tape punch) with blank paper tape before, between and after the files. The computer operator is then issued with instructions, via the console keyboard (ASR-print mode), to label the tape just produced.

The two object files are subsequently loaded in order to produce a core map which is then listed on the line printer. In this example, no attempt is made to run the program.



CARD DECK FOR JOB DAPEX FIG. 4 - 2

# 5 SYSTEM CONFIGURATION AND BUILDING

BOS may be supplied in one of the following ways:-

(1) As a complete system pre-configured on a moving head disc and a minimum core size of 16K. (Subsection 5.7)

(2) On a single magnetic tape which contains configurator routines to enable the user to configure his own BOS system on a moving head disc and a minimum core size of 16K. (Subsection 5.6)

(3) As a set of object tapes and self establishing subsystem tapes to enable the user to configure his own BOS system on a moving head or fixed head disc and a minimum core size of 12K. (Subsection 5.1)

Note: Some parts of the BOS Manual are meaningful only to the BOS user when BOS is configured from paper tape.

## 5.1 Configuration from Paper Tape

BOS is supplied as a set of object tapes and self-establishing subsystem tapes. The object tapes are loaded in a specified order together with a routine called B$SP -- System Parameter Program. B$SP must be written by the user for his particular system. It defines such things as the area of disc to be used by BOS, the disc block size to be used, etc. Once the BOS main system has been built by loading the relevant object tape using the linking loader (LDR-APM), a self-loading system tape of the system may be dumped out for future use should the system become corrupted.

When the main system has been built, it is initialised and it is then ready to run. During initialisation, the disc driver will be set up with the appropriate block size, the disc (if moving head) will be formatted appropriately, and the BOS main system will be written onto the disc. Loading of the required subsystems may proceed and the system is then ready for use. During this initialisation, a bootstrap tape (often called the 'RESTART' tape) may be punched if required. This can be used to restart the system should the high core-resident routine become corrupted.

## 5.2 BOS System Tape Generation

### 5.2.1 System Parameter Program (B$SP) Preparation

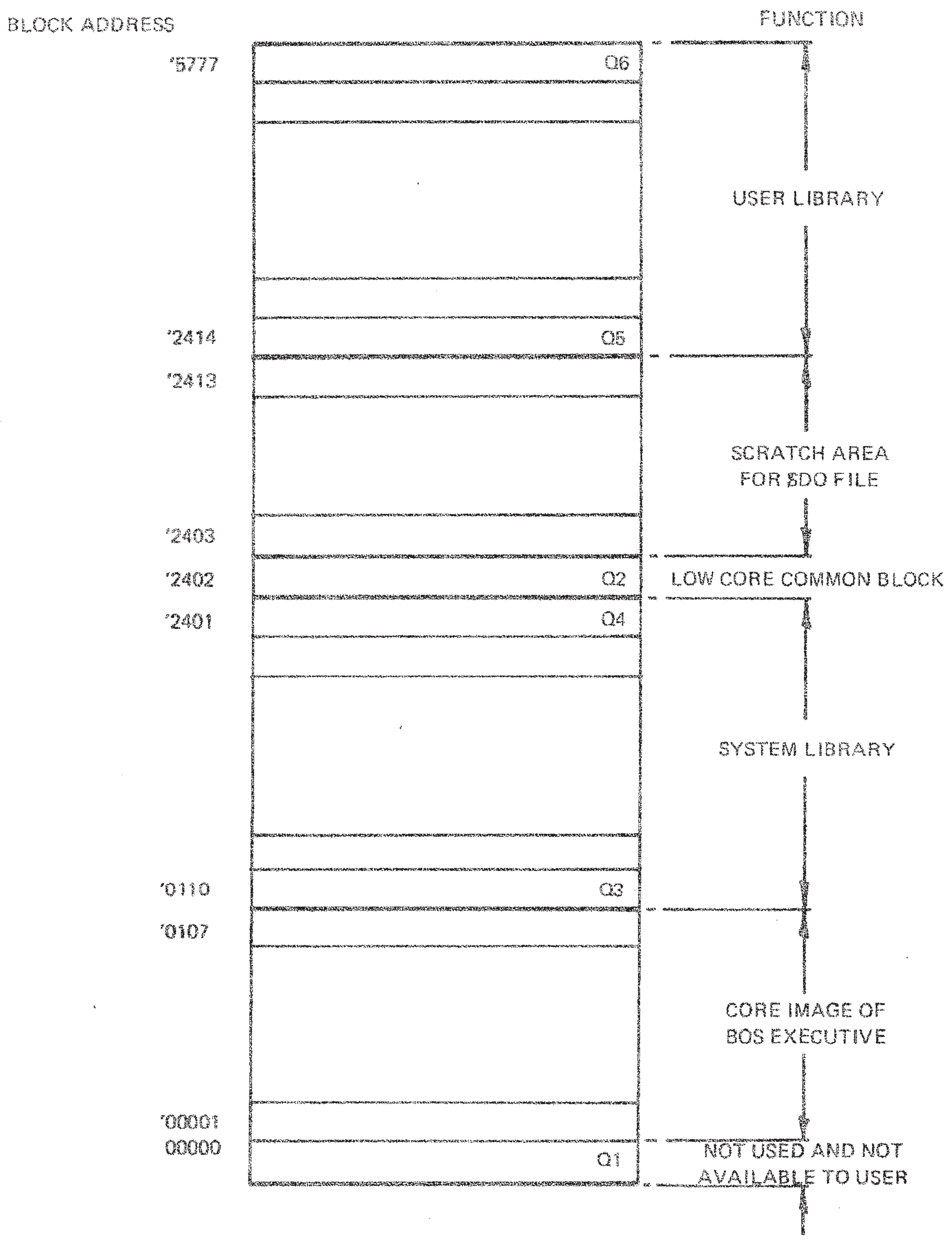
A program called System Parameter Program (B$SP) is required to generate the BOS SLST tape. The B$SP program is not supplied as a part of BOS because system requirements vary for each user. Therefore, each user must design B$SP to meet his own individual system requirements.

The components of B$SP are presented in Table 5-1. From inspection of the B$SP coding it can be seen that a number of variables concerning the system parameters must be supplied by the user. Variables for Q1 through Q6 are 32-bit constants and variables for Q7 through Q14 are 16-bit constants. Each of the variables is described to aid the user in determining the correct block addresses. Fig 5-1 is presented to illustrate a typical layout of the mass backing store and identify the variable block addresses. All block addresses are given in octal. Currently, BOS supports up to a maximum of 32767 (decimal) blocks.

Note: When calculating the number of blocks from the number of words divided by the block size, the resulting number of blocks must be rounded up to the nearest integer.

TABLE 5 - 1
B$SP CODING

| LOCATION 1 4 | OPERATION 6 10 | ADDRESS 12 | COMMENTS 30 |
|---|---|---|---|
| * |  | B$SP | SYSTEM PARAMETER PROGRAM |
| * |  |  |  |
|  | ABS |  | ABSOLUTE |
|  | ORG | '1000 | ORIGIN |
| * |  |  |  |
| STRT | CALL | B$IN | INITIALISE OPERATING SYSTEM |
| * |  |  |  |
| Q1 | OCT | < > | BLOCK ADDRESS OF START OF SYSTEM AREA |
| Q2 | OCT | < > | BLOCK ADDRESS OF LOW CORE COMMON BLOCK |
| Q3 | OCT | < > | BLOCK ADDRESS OF END OF SYSTEM LIBRARY |
| Q4 | OCT | < > | BLOCK ADDRESS OF START OF SYSTEM LIBRARY |
| Q5 | OCT | < > | BLOCK ADDRESS OF START OF USER LIBRARY |
| Q6 | OCT | < > | BLOCK ADDRESS OF END OF USER LIBRARY |
| Q7 | OCT | < > | MEMORY SIZE |
| Q8 | DXA |  | MODE IN WHICH MAIN SYSTEM OPERATES |
| Q9 | OCT | < > | AVAILABILITY OF LINE PRINTER (0 IF NOT) |
| Q10 | OCT | < > | BOOTSTRAP TAPE REQUIRED (0 IF NOT) |
| Q11 | OCT | < > | SYSTEM LIBRARY INITIALISATION REQUIRED (0 IF NOT) |
| Q12 | DEC | < > | NO OF TRACKS PER SURFACE |
| Q13 | DEC | < > | NO OF WORDS PER BLOCK |
| Q14 | DEC | < > | 0 IF NO FORMATTING NEEDED. 1 IF FORMATTING REQ'D. |
| Q15 | BSZ | 3 | FOR FUTURE ENHANCEMENTS |
| Q16 | BSZ | 2 | FOR FUTURE ENHANCEMENTS |
| * |  |  |  |
|  | END | STRT | END OF B$SP |
|  |  |  |  |



TYPICAL MASS STORE LAYOUT USING 128 WORD BLOCKS FIG. 5 - 1

Q1 This variable identifies the block address of the lowest part of the disc that is to be made available to BOS. For example, this number would be (0,0)if BOS begins at the lowest address on the disc. Q1 must be (0,0)if the Disc bootstrap is to be used. The Disc Bootstrap is always stored in absolute block 0.

Q2 This variable identifies the block address of the low core common save area (LCC). It immediately follows the System Library area. The LCC consists of one complete block.

Q3 This variable identifies the block address at the end of the system library (this block address is the lowest block address in the system library). As a rule of thumb, around 10,000 words will be occupied by the BOS Core Image, thus Q3 may be set accordingly. Alternatively, if a precise value is required to maximise disc usage, the variable should be set to (0,0). Then, the correct block address is inserted manually after the system has been completely loaded into memory. The procedure for evaluating the variable in Q3 is detailed in Appendix E.

Q4 This variable identifies the block address at the start of the system library. As a rule of thumb, the entire system library with full FORTRAN facilities and all subsystems occupies about 120,000 words.

Q5 This variable identifies the block address of the start of the user library, it also defines the area available for execution of $DO files. Each record of a $DO file requires 11 words but there can only be an integral power of 2 records held in one block. Thus the number of blocks required is calculated by first determining the largest power of 2 ($2^P$) x 11 word records that will fit in a block, and then dividing the total number of records required by this number, rounding up to the nearest block. e.g. for 128 word blocks, the largest number of 11 word records that can be held in the block is 11, thus the largest power of 2 records that can be held in the block is 8. Therefore a $DO area of 70 records will require $\frac{70}{8}$ blocks (rounded up) = 9 blocks.

Q6 This variable identifies the block address at the end of the user library, i.e., the last block on the disc allocated to BOS. Currently, the maximum value of this parameter is 32767.

Q7 This variable identifies the core size allocated to BOS (eg, '30000 for a 12K memory, '40000 for a 16K memory).

Q8 This variable identifies the mode in which the main system operates. It will contain a DXA or EXA instruction.

Q9 This variable indicates whether job or system error messages are to be output on the line printer as well as the ASR. It contains a 1 if a line printer output is required and 0 if a line printer output is not required.

Q10 This variable contains a 1 if a bootstrap tape is required and a 0 if a bootstrap tape is not required.

Q11 This variable contains a 1 if initialisation of the system library is required and a 0 if initialisation of the system library is not required.

Q12 This variable identifies the number of tracks which are available to BOS on each surface of the disc, except for the Honeywell moving head disc (4700 Series) when the number of disc surfaces must be specified.

Q13 This variable indicates the number of words (N) to be used for each block on mass store. The actual number of data words that are stored in one block is (N-3), the three additional words are used by BOS for block linking. There are certain restrictions on the choice of block size with different mass store devices and reference must be made to the listing of the disc driver for the disc being used.

Q14 This variable specifies whether the disc is to be formatted or not during the initialisation process.

For moving head disc this variable must be set:

= 1 if formatting required

= 0 if no formatting required

Normally, it will be set to 1 when the system is first set up on a 'virgin' disc, and then should the system become corrupt and have to be re-initialised, Q14 can be patched to zero to save time on initialisation. (Unless, of course, the disc format has become corrupt, in which case the disc must be reformatted).

For fixed head disc systems, Q14 should be one.

Q15
Q16 These variables are set to BSZ 3 and BSZ 2 pseudo-ops respectively and are reserved for future enhancement of BOS.

## 5.2.2 Loading the BOS Main System

The minimum core size in which BOS may be loaded from paper tape is 12K.

The BOS Main System will be supplied as an object library tape (B$MSYS) and a set of single object tapes which provide the user with the necessary options appropriate to his particular configuration. The procedures described below describe how to create an SLST for the system.

To load BOS main system from object paper tapes, the following procedure must be used. (Reference to the listing of LDR-APM should be made for instructions on the use of the loader).

Using the SLST and object versions of LDR-APM that are provided, relocate the loader so that it is situated from location '21000 upwards. The loader must be patched to allow loading above itself. Details of how to effect this change may be found in the loader listing.

Using LDR-APM, load the object tapes as follows:

(Note: If the core memory is greater than 16K, load in EXTENDED desectorising mode).

1. *Load B$SP.*

2. *Load B$MSYS – This tape contains several object modules but will appear to load continuously. The modules it contains are itemised in the B$MSYS library Tape Content listing.*

3. *If you have a card reader, load B$CR.*

4. *If you have a card punch, load B$CP (see Note 1).*

5. *If you have a line printer, load B$LP.*

6. *If you have magnetic tape, load B$MT.*

7. *Load your I/O library (see Note 2).*

8. *Load B$DD.*

9. *Load B$FT -- This must be the formatter routine (B$FT-) which corresponds to the disc being used.*

10. *If block size is less than or equal to 95 words, go to 11. Otherwise set a break point to 'X5000, load B$FP (Note 3).*

11. *Set a break point to 'X6000, load B$EF (Note 3).*

12. *Set a break point to 'X7000, load B$CO (Note 3)*

13. *Load appropriate disc driver (B$RB-).*

14. *Load B$TC.*

Loading complete (LC) should be printed on the ASR. Obtain a memory map for future reference.

Note 1:

For a card reader/punch both B$CR and B$CP must be loaded.

Note 2:

The I/O library tapes required depend upon the principal devices being used, as follows:

| | |
|---|---|
| ~~HSRP-IOL~~ | mandatory |
| MAG-IOL | mag. tape only |
| CRP-IOL | CR, or CR/P only |
| LP-FIOL | line printer only |
| GEN-IOL | mandatory |

GEN-IOL must be specially configured for BOS, refer to GEN-IOL listing for details.

Note 3:

| | |
|---|---|
| X = 2 | for 12K |
| X = 3 | for 16K |
| X = 5 | for 24K |
| X = 7 | for 32K |

If loading complete (LC) indication is not obtained after completing the BOS loading procedure, check to see if the latest revision of the I/O library has been used. If not, reload the BOS system using the latest revision of the I/O library.

After completing the above operations, load the PAL-AP subprogram into core starting at location '16000 and dump BOS into two segments of core using PAL-AP. The two segments of core to be used are defined as follows: Segment 1 consists of location '100 to location B$TF; and Segment 2 consists of location B$EF to location (*HIGH-1). These locations can be obtained from the loader map. These two SLSTs are the BOS main system. This tape(s) should be carefully preserved for future use should the BOS system become corrupted.

## 5.3 Initialising the BOS System

After generating a self-loading system tape(s) of BOS, the system is initialised using the following procedure. The switches which are referred to in the following procedure are located on the computer control panel.

1. *Turn on disc power.*
2. *Turn on line printer, if available.*
3. *Place the BOS self-loading system tape into the high-speed paper tape reader.*
4. *Depress MSTR CLEAR, set P-register to '1, set MA/SI/RUN to RUN, and depress START.*
5. *Move the tape to the second half of the system tape (or load system tape 2), depress MSTR CLEAR, set P-register to '1, set MA/SI/RUN to RUN, and depress START.*
6. *Set SENSE SWITCHES 1 and 2.*
7. *Depress MSTR CLEAR, set P-register to '1000, set MA/SI/RUN to RUN, and depress START.*
8. *The disc formatting routine will now be entered if formatting was specified. This routine sets up the disc driver for the appropriate block size and then, (in the case of a moving head disc) the disc will be formatted. This process can take some time on large discs. Only the area of disc specified for the BOS system to occupy will be formatted.*
9. *The BOS restart tape is generated by the high-speed punch. This tape should be preserved.*
10. *After a further delay, SYS LIB INITIALISED is typed on the ASR followed by START.*
11. *If the line printer is configured into the system, it will print SYSTEM START.*
12. *A question mark (?) is printed on the ASR.*
13. *Reset SENSE SWITCHES 1 and 2.*
14. *Execute a $JOB command.*
15. *The BOS subsystems can be loaded as described in Section 5.4.*

## 5.4 Loading BOS Subsystems

BOS subsystems are provided as self-establishing subsystem tapes. These tapes contain BOS commands and the text (in binary form) of the subsystem. The format of the tapes is:

```
$TYPE
Subsystem identification
$AT  BI,PR
$ES  <subsystem name>,BI,<start address>
Text of subsystem (binary)
$WAIT
$AT  CI,AK
```

A brief description of each subsystem available is given in Appendix B. To load the subsystems:

(1) Place a subsystem tape in the high-speed paper tape reader (the order of loading subsystems is immaterial).

(2) Attach command input to the paper tape reader ($AT  CI,PR).

(3) The subsystem identification will be typed out, and the subsystem will be loaded in. A '?' will then be typed out.

(4) If there are more subsystems to be loaded, place the next subsystem tape in the reader and type carriage return.

(5) Repeat step (4) until all subsystems are loaded. (Note: For 8K systems the FORTRAN subsystem ($FN) is not suitable.)

(6) To return control to the ASR keyboard type:   $AT  CI,AK
(Note: If the last subsystem tape loaded is left in the reader, typing a carriage return will cause the command input to be returned to the ASR by the last command on the subsystem tape.

Details of the contents of each subsystem can be found in the Preconfigured Tape List for each subsystem.

## 5.5 Generating the System Library ($SL)

The system library comprises a set of standard subroutines in object code which can be called and linked to users' programs at load time by including $SL on the parameter in the LOAD command (see Section 3.22). The library can be configured to a user's particular requirements.

To generate SSL, the normal procedure is to establish a number of object files containing all the required object modules, and then create $SL using the 'NGROUP' command. In the following procedure the following notation is used to indicate that an object file is to be established:

```
ES <file name> = <object tape name>
```

All files are established by attaching object input to the paper tape reader ($AT  OI,PR) and using the establish command ($ES  <file name>,OI).

(1) Do you use FORTRAN? – If *not*, go to step (5).

(2) Do you have a Line Printer?

| | | | |
|---|---|---|---|
| *Yes* | – | ES | F1 = F$TR for line printer |
| *No* | – | ES | F1 = F$TR for ASR |

(3) ES F2 = FIO-VAR

(4) Do you want normal source type output on the paper tape punch rather than 'listing' output? (For 'listing' type output, the first character of the record is interpreted as a pagination control character, whereas for normal source output the first character of the record is punched normally.)

| | | |
|---|---|---|
| LISTING | — | go to step (5) |
| SOURCE | — | ES F3 = F$W2 |

(5) ES F4 = ASR33-IOL (if ASR 33 is used)
or
ES F4 = ASR35-IOL (if ASR 35 is used)

(6) ES F5 = HSRP-IOL

(7) Have you a CR or CR/P?

| | | |
|---|---|---|
| *No* | — | go to step (8) |
| *Yes* | — | ES F6 = CRP-IOL |

(8) Have you a line printer?

| | | |
|---|---|---|
| *No* | — | go to step (9) |
| *Yes* | — | ES F7 = LP-FIOL |

(9) ES F8 = BOS-IOL

(10) Have you magnetic tape?

| | | |
|---|---|---|
| *No* | — | go to step (11) |
| *Yes* | — | ES F9 = MAG-IOL |

(11) ES F10 = GEN-IOL (Note: GEN-IOL must be specially configured for BOS as described in the GEN-IOL listing).

(12) Establish files for all maths subroutines required using the appropriate maths library object tapes supplied.

(13) Establish ULIB and accumulators.

(14) To create $SL, use the NO GROUP command to create a single file from all the files you have established, e.g.:

```
$NG $SL=F1,F2,F3,F4,F5,F6,F7,F8,F10,....,F15
```

(15) Delete all the files (F1,F2,....,Fn) that you established to create $SL.

Your system library is now ready, and the BOS system is fully configured for use.

Note: Any user written subroutines may be included in or added to the system library.

## Adding Subroutines to the System Library

Adding subroutines to an already existing system library may be done as follows:

(1) Establish object files of subroutines to be added to the library, e.g., $F1,$F2,$F3....

(2) Make a new system library using the command:

```
$NG  $NSL=$SL,$F1,$F2,$F3,....
```

(3) Delete the old library:

```
$DE  $SL
```

(4) Change the name of the new library:

```
$NA  $NSL=$SL
```

## 5.6 Configuration from Magnetic Tape

### 5.6.1 BOS Initialisation

BOS is supplied as a single magnetic tape which should be loaded onto unit 1 in Protect mode and positioned at BOT. The Disc Key-in-Loader should be stored manually into the computer from the control console (see Appendix G). The Magnetic Tape Key-in-Loader should be stored manually from core location '34001 (see Appendix G).

Now the following procedure is performed:

(1) Switch on all BOS peripherals.

(2) Set SI mode and Master Clear.

(3) Set Sense Switch 1.

(4) Set P = '34001.

(5) Set computer in RUN mode.

(6) Press START

The Magnetic Tape Bootstrap will be read into core. This bootstrap is the first file on the magnetic tape and is capable of reading BOS binary files into core. The bootstrap is used to read the next file (the initial BOS configurator, B$CONFIG-1) from magnetic tape. This initial configurator prints a start message on the ASR and asks a number of questions in conversational mode (see Subsection 5.6.3). When an acceptable answer to each question has been typed in by the user the configurator goes on to the next relevant question. The questions asked by the configurator vary according to the devices specified.

When the questions have been answered correctly, the configurator chooses and reads the correct BOS system from the magnetic tape into core according to the options specified. The BOS system contains (and is linked to) the second configurator (B$CONFIG-2) which is entered directly. This linked configurator sets up the correct paper tape parity routines and card codes, sets the number of lines per page for the line printer, and calculates the answers to the BOS configuration questions in B$SP. The BOS initialisation routine (B$IN) is then entered at '1000 and BOS initialises itself. The disc driver is set up for the appropriate block size and the disc is formatted if formatting was specified. The restart tape is punched if requested. The system library is initialised (if initialisation was specified), and the message SYS LIB INITIALISED is typed on the ASR followed by START. If the line printer is configured, the message SYSTEM START is printed on it. Finally a question mark is typed on the ASR to request input from the keyboard.

## 5.6.2 Subsystem and System Library Configuration

At this point Sense Switch 1 should be reset and the following commands should be typed: -

```
$JOB CONFIG

$AT SI, M1

$ES $CONF3, SI

$DO $CONF3
```

$CONF3 is the subsystem and system library configurator. This command file is now obeyed.

The configuration subsystems are read from magnetic tape and a message is printed on the ASR to indicate the start of the BOS subsystem configuration. The required BOS subsystems are established from magnetic tape and, when this process is complete, the message "END SUBSYSTEM CONFIGN" is printed on the ASR. The required system library objects are now established from magnetic tape and, when the library has been created, the message "END SYST LIB CONFIGN" is printed on the ASR. BOS is now ready for use.

If an error occurs during the above procedure a BOS error mnemonic followed by the message "JOB ABORTED" are printed on the ASR'

## 5.6.3 Question and Answer Sequence

This is as follows:

| Question | Answer |
|---|---|
| (1) Core Size | 16, 24 or 32 x1024 words of core |
| (2) R.P.G. | YES (sets 95 words/block) or |
| | NO (sets 443 words/block) |
| (3) High speed paper tape reader | PR or NO |
| (4) High speed paper tape punch | PP or NO |
| (5) Line printer | LP or NO |
| (6) Card reader or reader/punch | CR or CP or NO |
| (7) Magnetic tape (7 track) | M7 or NO (see note in Subsection 5.6.5) |
| (8) Card codes | PD for Series 16 peripheral device code |
| | ED for EDP-compatible code |
| | EB for EBCDIC code |
| (9) Punch parity | F8 for forced 8 parity |
| | EV for even parity |

| Question | Answer |
|---|---|
| (10) Paper tape bootstrap | YES OR NO |
| (11) Reader parity | F8 for forced 8 parity |
| | EV for even parity |
| | NO for no parity check |
| (12) System messages on line printer | YES or NO |
| (13) Number of lines per line printer page | 37 or 55 |
| (14) Fixed or variable FORTRAN device numbers | FI for fixed (not supported) |
| | VA for variable |
| (15) High Speed arithmetic | YES or NO |
| (16) Number of disc units | 1 to 4 |
| (17) Number of disc surfaces per unit | 2 to 20 |
| | (total number of blocks allowed is 32767) |
| (18) Percentage of disc to be allocated for use by BOS | 1 to 100 |
| (19) Size of system area as percentage of total number of disc blocks allocated to BOS | 1 to 100 |
| (20) Disc formatting | YES or NO |
| (21) System library initialisation | YES or NO |

Note: Disc formatting and system library initialisation must both be specified when BOS is created for the first time on a new disc.

### 5.6.4 Changing the BOS configuration updating or restoring BOS Without Losing Files

The procedure is exactly as described in Subsection 5.6.1. To preserve all files on the disc, disc formatting and system library initialisation must not be requested. Any changes to the BOS configuration may be made by answering the configuration questions in the desired manner (Subsection 5.6.3). Answers to questions (1), (2), (16), (17), (18) and (19) must not be changed if files are to be preserved.

Changes may be made to the devices supported, the card code, parity routines, system messages required or not on line printer, and number of lines per line printer page.

When configuration is complete the message START is printed on the ASR, SYSTEM START on the line printer if line printer messages are configured and, finally, a question mark is printed on the ASR. Sense Switch 1 should be reset at this point.

The BOS is now ready for use and all system and user files have been preserved. If the user files are not required a $JOB command may be typed.

## 5.6.5 BOS Disc Bootstrap Restart Procedure

When BOS is configured from magnetic tape or supplied on a pre-configured disc the following restart procedure is applicable:-

(1) Set SI mode

(2) Master clear

(3) Set P = 1

(4) Set RUN mode

(5) Start

The message RESTART is printed on the ASR followed by a question mark. In addition if the system is configured for system messages on the line printer, a message is printed on the line printer.

If this procedure does not work, check the Disc Key-in-Loader and try again if appropriate. If the system still fails to start, the procedure for restoring BOS from magnetic tape (Subsection 5.6.4) should be tried.

This restart procedure may be used if BOS has been configured from paper tape provided that:

(1) BOS is configured from disc block 0

(2) A moving head disc is used

(3) A Disc Key-in-Loader is used instead of the Paper Tape Kay-in-Loader

Note: If the BOS system to be built has no magnetic tape, a two-stage building procedure is required. The first stage build follows the procedure in section 5.6.3 but question 7 must be answered "M7" to enable the configurator to operate initially. This must then be followed by a second stage build using the procedure in section 5.6.4 but answering question 7 with NO.

## 5.7 Configuration from Pre-Configured Disc

BOS is supplied as a single moving head disc complete with all subsystems and system library. The Disc Key-in-Loader should be stored manually into the computer from the control console (see Appendix F). Set SI mode, master clear, set P = 1, set the computer in RUN mode and press start. The message RESTART is printed on the ASR, a message is printed on the line printer if the line printer is configured followed by a question mark on the ASR. A $JOB command should be executed and the system is ready for use.

# 6 OPERATING BOS

This section describes some of the special procedures and facilities that are useful when operating BOS. The normal operating commands and their format, and use, are fully described in Section 3.

## 6.1 Restarting BOS

WARNING: On no account should the computer be stopped by switching into the 'Single Instruction' mode while a subsystem OR user program which establishes files is executing. This will cause corruption resulting in loss of all files in the User and possibly System Libraries.

### 6.1.1 Restarting During Execution of a Subsystem or User Program

Two methods may be used to interrupt the running of a subsystem or user program under BOS. The current operation will be aborted.

(1) Press the START button – this causes an interrupt and results in the message INT being typed on the ASR.

Command Input (CI) will be attached to the ASR keyboard.

(2) Unsolicited key-in – at certain times, BOS may be interrupted by typing any character on the ASR even though BOS has not asked for any input from the ASR. This results in the message UNS being typed on the ASR and Command Input (CI) will be attached to the ASR keyboard.

Note: These two methods will only work if the BOS core resident routine in high core and the low core common area (up to '77), have not been corrupted.

### 6.1.2 Restarting BOS Manually

BOS can be restarted manually by using the following procedure unless the top sector of memory has been overwritten.

1. *Set MA/SI/RUN to SI.*
2. *Depress MSTR CLEAR.*
3. *Set P-register to 'X7000, set MA/SI/RUN to RUN, and depress START (see Note 3. Section 5.2.2 for definition of values of 'X').*
4. *The ASR types RESTART.*
5. *The command input (CI) is attached to the ASR and a question mark (?) is typed.*
6. *The user can then type control commands.*

If the above procedure does not work, the following courses of action can be taken:

either (1) *Load BOS restart tape into the high-speed paper tape reader, set P-register to '1, set MA/SI/RUN to RUN, depress START.*

or (2) *Load the BOS self-loading system tapes, then –*

(a) *Set sense switch 1*

(b) *Reset sense switch 2*

(c) *Manually set the contents of Q10, Q11 and Q14 to zero*

(d) *Set P to '1000*

(e) *Depress START*

Should this operation not work, the disc has probably become corrupted, thus precluding any form of restart. In this case, the BOS system must be re-initialised using the BOS binary system tapes generated when the system was built. Refer to Section 5.3 for the initialisation procedure. Files on the disc will be lost during this operation.

## 6.2 Error Notification and Action

Three types of errors may be notified to the operator via the ASR:

| | | |
|---|---|---|
| (1) | User errors: | Message format 'UE XX' |
| (2) | FORTRAN errors: | Message Format 'FE XX' |
| (3) | System errors: | Message format 'SE XX' |

A list of error codes used by BOS main system is given in Appendix 'A', however, system errors may be generated by other parts of the system, and these will not necessarily appear in this list.

### 6.2.1 System Error Marker

Errors will normally result in the system error flag being set, the current operation will be aborted, and return will be made to system mode where BOS will solicit the next command. However, when the system error marker is set, all commands except:

```
$ATTACH
$JOB
$MREWIND
$DO
$WAIT
$RESET
$TYPE      )    these will cause the next record read on the command input stream
$NEXT      )    to be ignored
```

are skipped.

The $RESET and $JOB command will reset the error marker and allow normal command decoding to continue.

An exception to this is when an error occurs and command input is currently attached to the ASR keyboard. In this case, an automatic reset is executed and there is no need to type in a $RESET command.

## 6.3 Use of Sense Switches

The sense switches on the control panel can be used to control certain functions of the system:

| | | |
|---|---|---|
| Sense Switch 3 | SET | Ignore all User and FORTRAN run time errors. The error routine will return directly to the caller without setting the system error flag, or outputting an error message on the ASR.<br><br>Note: System Errors cannot be suppressed. |
| Sense Switch 1 | SET | Halt on all System, User or FORTRAN run time errors. If SS3 is reset, on pressing the START button, the system will continue by setting the system error flag, typing the error message and returning control to BOS. If SS3 is set, on pressing the START button following User or FORTRAN errors, return will be made directly to the calling program (this does not apply to System Errors). |
| Sense Switch 2 | SET | This is only used during system initialisation (see Section 5.3). |

# 7 FORTRAN PROGRAMMING FOR BOS

BOS provides the FORTRAN programmer with a number of extended facilities for program control, and disc file accessing. This section gives the statements that are available to him, and a description of their use.

It should be remembered that when a subroutine argument is specified, in all cases, either a variable name may be specified, or the argument may be explicitly defined as a literal. For example, several statements call for file names as arguments. The argument given can either be the name of a 6 character (3 word) variable which contains (or will contain) the file name, e.g.:

```
DOUBLE PRECISION FN
DATA FN/6HFILE99/
-
-
CALL B$FN7 (FN)
```

or the file name can be specified explicitly, e.g.:

```
CALL B$FN7 (6HFILE99)
```

The former example enables the file name to be changed at run-time. *File names must contain exactly 6 characters. Trailing blanks must be added to names if necessary.*

## 7.1 Program Control

### 7.1.1 Return to Operating System

| | |
|---|---|
| CALL EXIT | Terminates execution of the program and causes control to be returned to the Operating System. |
| | Note: This statement should be used in place of the normal FORTRAN 'STOP' statement which will cause a system error. |

### 7.1.2 Error Return to Operating System

| | |
|---|---|
| CALL ERROR (2Haa) | This causes the two character error code 'aa', specified by the argument to be printed on the ASR, and the current job to be aborted. |

### 7.1.3 Start Execution of Another Program

| | |
|---|---|
| CALL CHAIN (file name) | This causes the binary file whose name is specified by the 6 character argument to be read into core and executed from its *START address (see LOAD command, Section 3.22). The file specified must be a binary file. Data may be passed between the programs by use of FORTRAN COMMON. (See Section 3.22.3 on setting the COMMON base address when loading.) |

## 7.1.4 Overlay Programs

Two statements are provided to enable a segment of a program to call another segment stored as a binary file on the disc, and subsequently to return to the first segment at a point immediately following the call. This facility differs from CHAIN in that the breakpoint in the calling program is recorded so that return can be made to that point.

| | |
|---|---|
| CALL SEGMNT ($segname_1$) | Causes the address of the next sequential statement in the current segment to be saved, and the binary file specified by the file name '$segname_1$', to be loaded and executed from its normal start address. |
| CALL RETSEG ($segname_2$) | Causes the current segment to be terminated, and the binary file specified by the file name '$segname_2$' to be loaded. Execution of $segname_2$ will resume at the next sequential statement after the last CALL SEGMNT statement executed by $segname_2$. If '$segname_2$' has not previously executed a CALL SEGMNT statement, execution will start at its normal start address. |

Note: (1) Variables must be passed from one segment to another via FORTRAN COMMON. (See Section 3.22.3 on setting the COMMON base address when loading).

(2) When using this system the user must ensure that B$CO, B$EF and B$FP are not corrupted in any way at any time. The routines are normally located in the top three sectors of memory occupied by the BOS system.

Since COMMON normally overwrites B$EF and may also overwrite B$FP, if COMMON is used, the common base must be set at load time by a *C=XXXXX parameter for COMMON base addresses set by the BOS loader or at FORTRAN translation time (see Section 3.18) for COMMON base addresses set by the FORTRAN translator. For 95-word BOS systems on

| | | | |
|---|---|---|---|
| 8K, | XXXXX | = | 15777 |
| 12K, | XXXXX | = | 25777 |
| 16K, | XXXXX | = | 35777 |
| 24K, | XXXXX | = | 55777 |
| 32K, | XXXXX | = | 75777 |

For BOS systems using block sizes > 95 words on

| | | | |
|---|---|---|---|
| 8K, | XXXXX | = | 14777 |
| 12K, | XXXXX | = | 24777 |
| 16K, | XXXXX | = | 34777 |
| 24K, | XXXXX | = | 54777 |
| 32K, | XXXXX | = | 74777 |

(3) A segment, re-entered via a CALL RETSEG (SEGNAME) statement, is entered with all local variables initialised as if it had been entered via a BOS $EX command. If variables are to be preserved through a CALL SEGMENT, CALL RETSEG sequence, they must be in common, eg the following program is illegal.

```
      DO 1 I = 1, 10
      CALL SEGMNT (6HSEGMTZ)
1     CONTINUE
      CALL EXIT
      END
```

The legal version of this program is

```
      COMMON I
      DO 1 I = 1, 10
      CALL SEGMNT (6HSEGMTZ)
1     CONTINUE
      CALL EXIT
      END
```

(4) The CALL NAMECH (FILE 01, FILE 02) statement must not be used in any part of a segmented program;

(5) The CALL SEGMNT (SEGNAME) statement is not allowed in a subroutine.

(6) No part of a segmented program may be executed by a loader *E command (See Section 3.22.5).

(7) Each segment entered or re-entered by a CALL SEGMNT or CALL RETSEG statement is entered with interrupts enabled.

Example of Overlay Program:

Consider a FORTRAN program written as three segments. These segments are held on the BOS disc in binary files named SGMNT1, SGMNT2 and SGMNT3 respectively (see Fig. 7 - 1).

SGMNT1 is loaded and executed. It then calls SGMNT2 which overlays SGMNT1. SGMNT2 calls SGMNT3 which overlays SGMNT2.

SGMNT3 returns to SGMNT2 which is reloaded, and execution resumes at statement number 20 in SGMNT2. SGMNT2 then returns to SGMNT1 which is reloaded and execution resumes at statement number 10 in SGMNT1.

### 7.1.5 Fetch Current Program Name

| | |
|---|---|
| CALL MYNAME (file name) | This statement causes the binary file name of the currently executing program to be copied into the 6 character argument specified by (file name). |

## 7.2 **Disc File Access -- Sequential Files**

The following set of statements are provided to enable the FORTRAN user to access sequential disc files. Files are referenced by using device numbers 7, 8 and 9 in READ/WRITE and control statements. Up to three sequential files may be open simultaneously.

### 7.2.1 Define File Name

| | |
|---|---|
| CALL B$FNn (file name) | This statement associates the disc file specified by (file name) with device number 'n'. e.g., CALL B$FN7 (6HFSEVEN) this statement causes device 7 to be associated with a file called 'FSEVEN'. |

```
SGMNT1

         -
         -
         CALL   SEGMNT (6HSGMNT2)
    10   -
         -
         -
         -
         -
         END
```

```
SGMNT2

         -
         -
         CALL   SEGMNT (6HSGMNT3)
    20   -
         -
         -
         CALL   RETSEG (6HSGMNT1)
         END
```

```
SGMNT3

         -
         -
         -
         -
         CALL   RETSEG (6HSGMNT2)
         END
```

EXAMPLE OF OVERLAY PROGRAM FIG. 7 - 1

Notes:

(1) If B$FNn is not called before a file is accessed, the following default file names will be assumed:
Device No. 7 - FILE07
Device No. 8 - FILE08
Device No. 9 - FILE09

(2) If 'n' is not specified (i.e., CALL B$FN (file name) device 9 will be assumed.

## 7.2.2 Input/Output Statements

```
READ   )
       ) (u, f) list
WRITE  )
   or
READ   )
       ) (u) list
WRITE  )
```

Normal formatted (or unformatted) I/O statements are used. The first time an I/O statement specifying a file device (u = 7, 8 or 9) is executed, the following action will occur:

On READ - the file whose name is associated with the device number specified will be opened for *input*. If the file name is not in the dictionary, a BOS 'ON' error will occur and the job will be aborted. Any attempt to WRITE to a file which has been opened for input will result in a BOS 'IM' error and the job will be aborted.

On WRITE - the disc file directory is searched for the file name associated with the specified device, and if it is not found a new file is opened with the given name. If a file of the same name *is* found, a BOS 'ED' error occurs, and the job will be aborted. Any attempt to READ from a file opened in output mode will result in a BOS 'OM' error and the job will be aborted.

Note: If B$FNn has not been called before the first READ/WRITE operation, the default file names will be used.

## 7.2.3 File Control

```
REWIND    n)
ENDFILE   n)
```

These statements are synonymous and cause the following actions to the file associated with device n. The file is closed, and it is reset so that a subsequent read from this device will cause the first record in the file to be read. A file opened in output mode by a WRITE statement may be read after a 'REWIND' has been executed on that file.

```
BACKSPACE n
```

This command is treated as illegal on disc files, and the FORTRAN error 'ID' will result (ILLEGAL DEVICE).

## 7.3 Disc File Access -- General Facilities

### 7.3.1 Deleting Files

| | |
|---|---|
| CALL B$DFn | This will cause the file whose name is associated with device number n (n = 7, 8 or 9) to be deleted. If B$FNn has not been called before execution of this statement, the default file name will be assumed (see Section 7.2.1). |
| CALL B$DNF (file name) | This will cause the file whose name is specified by (filename) to be deleted. For example, CALL B$DNF (6HDUMMYF) will cause the file named 'DUMMYF' to be deleted. CALL B$DNF (IFN) will cause the file whose name is contained in the variable named IFN to be deleted. Note: In both these statements, should the specified file not be found, a BOS 'DN' error will occur. |

### 7.3.2 Search for Named File

| | |
|---|---|
| CALL B$SNF (file name, n) | This causes the disc file directory to be searched for a file whose name is specified by (file name). The value of the integer variable n will be set: = 1 if the file is found = 0 if the file is not found For example, CALL B$SNF (6HFILE77,I) CALL B$SNF (IFN,M) (where IFN is a variable containing the file name) |

### 7.3.3 Change Name of File

CALL NAMECH (file name 1, file name 2)

This causes the name of the file specified by (file name 1) to be changed to the name specified by (file name 2). A check is made for the existence of a file with the first name, and the non-existence of a file with the second name. If both these conditions are true, the name of the file is changed to the new name. If the first check fails, a BOS 'ON' error occurs. If the second check fails, a BOS 'ED' error occurs. For example,

```
CALL NAMECH (6HFILE01,
             6HFILE02)
```

CALL NAMECH (IFN1, IFN2)
(where IFN1 and IFN2 are variables containing the two file names)

Note: A system file cannot be renamed as a user file and vice-versa, i.e., (file name 1) and (file name 2) must both start with either a $ (dollar sign) or with a letter. This statement must not be used in any segment of a segmented program.

## 7.4 Disc File Access - Random Files

Facilities exist which enable a random file to be defined or referenced. Any record within the file can be directly accessed by specifying the number of the record required. Records within the file are of fixed length and are numbered consecutively through the file starting from 1. It should be noted that a random file is not compatible with a BOS Sequential File, although clearly a random file itself can be referenced sequentially. The normal BOS SOU command can be used with a formatted random file only and, where the record length exceeds 60 words, each record will be output in multiples of 60 word records. Unformatted random files are only accessible to FORTRAN programs.

### 7.4.1 Defining and Referencing a Random File

```
CALL DEFINE          (File name, file number, nre, lre, fm, rp)
```

| | |
|---|---|
| where: file name | specifies the name of the file being defined or referenced. This is a normal BOS file name. |
| file number | an integer variable or constant which defines a "file number" which will be used to reference the given file. This value must be unique in relation to all files referenced in a given program or overlay and this value will be associated with the given file in respect of all further uses of this file number in the given program or overlay. |
| nre | an integer variable, constant or expression defining the maximum number of records in this file. The maximum number of records is limited to 32767 unless the record size exceeds the (block size - 3). See note 1. |
| lre | an integer variable, constant or expression defining the length of each record in words. The maximum length is 500 words (Note that 1 word = 2 characters). |
| fm | the hollerith constant U for unformatted or F for formatted, or an integer variable containing the hollerith constant U or F left justified. |
| rp | an integer variable which will be used to define the record to be accessed in this file. Initially, this will be set to 1 and will be updated each time the file is accessed to indicate the next available record. It may be set by the program to indicate which record to access next time the file is referenced. |

The above statement will cause the file specified to be either created on the BOS disc as an empty formatted or unformatted file or vetted against the existing file for compatibility. To make the records of the file readily accessible, an indexing technique is used for the file.

Up to 75 files can be defined in any one program or overlay. Of these 75 files, full details are recalled for the last five files referenced providing quick access. The introduction of a sixth file would necessitate recalling its details in place of those of the least used of the five files recalled. In general, this feature can give the user optimum speed in referencing random files.

Any program or overlay referencing a random file must contain a define statement for that file. If an attempt is made to reference an undefined random file a BOS 'ON' error will occur.

### 7.4.2 Input and Output

READ (0, f) n, list

READ (0) n, list

WRITE (0, f) n, list

WRITE (0) n, list

where: the unit is always zero for random files only; f is the format statement label for formatted files only; n is an integer variable specifying the file number associated with the file to be referenced. This is always the first item in the list; list is the normal FORTRAN Input/Output list.

An attempt to reference an unformatted random file with a formatted READ/WRITE statement (or vice-versa) may result in a BOS 'RT' error. On a READ/WRITE request, the record specified by the integer variable associated with the file will be accessed.

It should be noted that with random files there is no restriction on access mode, the file may be read from and written to. When the list specifies data greater in length than one record, for formatted READ/WRITE a '/' character must appear in the FORMAT statement in the appropriate place otherwise the excess data will be lost, whereas for unformatted READ/WRITE the next record is automatically referenced without loss of data.

### 7.4.3 Auxiliary Input/Output Statement

| | |
|---|---|
| BACKSPACE | 0 |
| ENDFILE | 0 |
| REWIND | 0 |

These statements should not be used as their action is not relevant to a random file. Accordingly, if used a FORTRAN Run-time error 'ID' (illegal device) will result.

### 7.4.4 Errors

The following errors can occur and they are reported as FORTRAN errors. System errors which can occur carry the normal BOS definition.

| | | |
|---|---|---|
| FE | ID | Illegal device. This is a general error which can include misuse of the auxiliary input/output statements for device 0 (see Subsection 7.4.3 above). |
| FE | RA | Random File range error either record size greater than 500 words or maximum number of records in excess of upper limit or record to be accessed outside bounds for file. |
| FE | RC | Either definition of random file incompatible with existing random file or attempt to define too may random files or use of random file inconsistent viz. formatted use with unformatted file or vice versa. |
| FE | RN | Either attempt to reference an undefined random file or a non-random file for random access. |

Note 1

When the record size exceeds the (block size - 3), the maximum number of records is

$$\left[ \frac{32767 \times (\text{block size} - 3)}{\left(\frac{\text{Record size}}{60}\right)_{RU} + \text{record size}} \right]_{RD}$$

where RU means rounded up to the integer above
RD means rounded down to the integer below.

## 7.5 FORTRAN PAUSE and STOP Statements

A FORTRAN PAUSE statement results in communication with the operator, using the ASR. Continuation enables a parameter to be returned. Full details are contained in the relevant B$UE listing.

A FORTRAN STOP statement returns control to BOS giving the FORTRAN run-time message 'ST' to identify the statement actioned.

# 8 DAP-16 PROGRAMMING FOR BOS

For the programmer writing programs in DAP-16, BOS offers a number of useful facilities. These include file handling capability and various returns to the operating system which enable the job to continue. Although it is quite possible to run most DAP-16 programs under BOS, it is felt that to get the best use out of the system, the DAP-16 programmer should have more detailed information about the structure and operation of the operating system, and how data is handled and organised by the system.

The first part of this section (8.1 to 8.4) deals with the structure and operation of BOS, the second part of the section (8.5) gives more details of the use of BOS facilities by the DAP-16 programmer.

It is assumed that the reader is familiar with the configuration of the BOS system, as reference is made to the various modules making up the system e.g. B$SP, B$CO, B$EF. The purpose of these modules is described in the relevant listings. (See also section 5.)

## 8.1 BOS Executive Organisation

### 8.1.1 System Disc Storage

The core-image of the entire BOS executive is stored on the disc. The core image is written in blocks of N words where N is the block size of each disc block as defined by the user in Q13 of B$SP (see section 5.2.1). Data is written in sequential blocks without string pointers, i.e. all N words are used for data. All disc addresses used, are relative to the starting block address of the system disc area as defined by the user in Q1 of B$SP (see section 5.2.1).

### 8.1.2 Core Layouts

The BOS executive in core is divided into 4 parts.

(1) The BOS core resident section occupies the top sector of core and consists of the routine B$CO and the relevant disc driver. This section is punched in 8-8 format on the BOS bootstrap tape and must be preserved in core at all times if return to BOS control is desired. The section is also written onto the disc starting at relative disc address 1. It will occupy a number of sequential disc blocks which will depend on the disc block size.

(2) The BOS routine B$EF is stored in core one sector below B$CO. This routine enables files to be read, written, deleted, etc., and may be overwritten by a user program if desired. The sector containing B$EF is written on the disc starting from the next relative disc address above the last block used by B$CO. This sector is written back into core from disc every time BOS returns to system mode.

(3) The BOS main system occupies core from location '100 to the address of B$TM (top of main system). It is written on the disc starting from the next relative disc address above the last block used by B$EF. This section of core may be overwritten by a users program as it is written back into core every time BOS returns to system mode.

(4) The BOS low core common block is the area of core between location '22 and '77. This is written onto the disc at the relative disc address defined by the user in Q2 of B$SP (see Section 5.2.1). This block in core may be overwritten by a user's program provided:

(a) The user's program did not establish or delete any BOS file, and

(b) Return to BOS control is via a manual or bootstrap restart.

### 8.1.3 File Pointer Tables

In addition to the core usage as described in section 8.1.2 BOS requires, in core, the following 3 file pointer tables (also see section 8.2.2):

(1) The main table is situated immediately below B$EF and is called B$FP.

(2) The second main system table starts from location B$TM (top of main system) and extends in ascending core locations up to B$TM + N + 9 (where N is the block size).

(3) A private pointer table is required by $DO commands. This 'floats' on top of the second system table (2) and extends a further N + 9 locations upwards in core.

## 8.2 BOS File Structure

BOS writes files consisting of a number of chained blocks on the disc. This chaining scheme allows the files to be written in blocks anywhere on the disc. It also allows simple garbage collection for files that are no longer needed. (See Figs. 8 - 1 to 8 - 4.)

The available disc area is split into 5 sections by BOS (see also Fig. 5 - 1).

(1) An area containing the core image of the BOS executive.

(2) An area containing the system library which consists of files which are to be preserved from job to job.

(3) A single block containing the BOS low core common area.

(4) A scratch area used by $DO commands.

(5) An area containing the user library which consists of files which are not to be preserved from job to job.

These sections are structured as follows:-

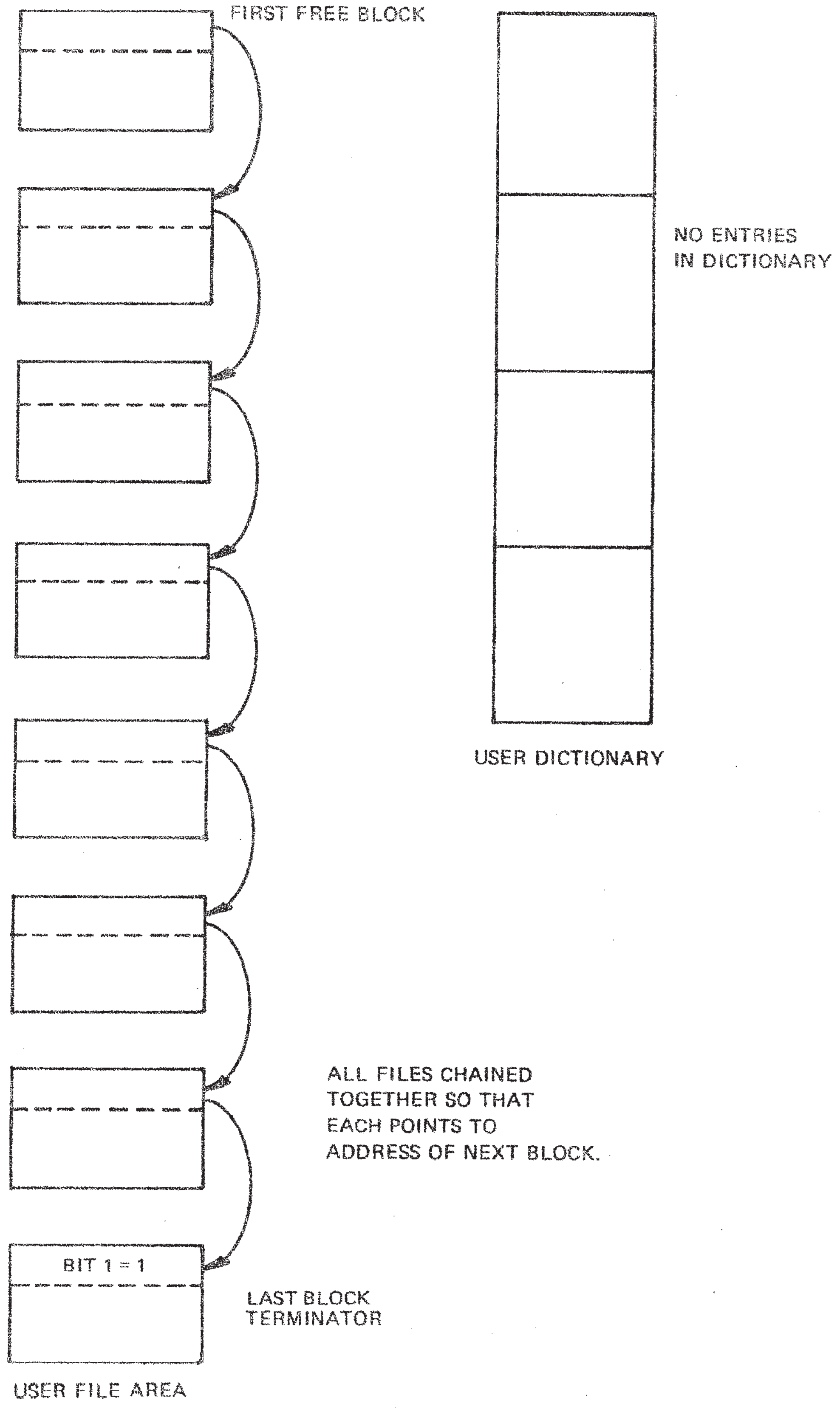
(1) The core image of BOS is written in a series of blocks with increasing block addresses from the block address specified in B$SP at the start of the system disc area (usually block 0000) up to the block one below the block defining the end of the system library.

(2) The system library is the area defined by the parameters Q3 and Q4 in B$SP (see section 5.2.1). The area is initially set up as 2 parts.

   (a) The first block of the system library is defined as the first block of the system dictionary (this is the block with the highest block address).

   (b) The second block of the system library is defined as the first system free block (this is the block with the second highest block address). All other blocks in the system library are linked together in order of descending block addresses. The string or chain so formed is known as the system free string.

(3) The single block defined as the low core common block contains the core image of the BOS low core common area (Location '22 – '75).



USER FILES AFTER $JOB COMMAND INITIALISATION FIG. 8 - 1

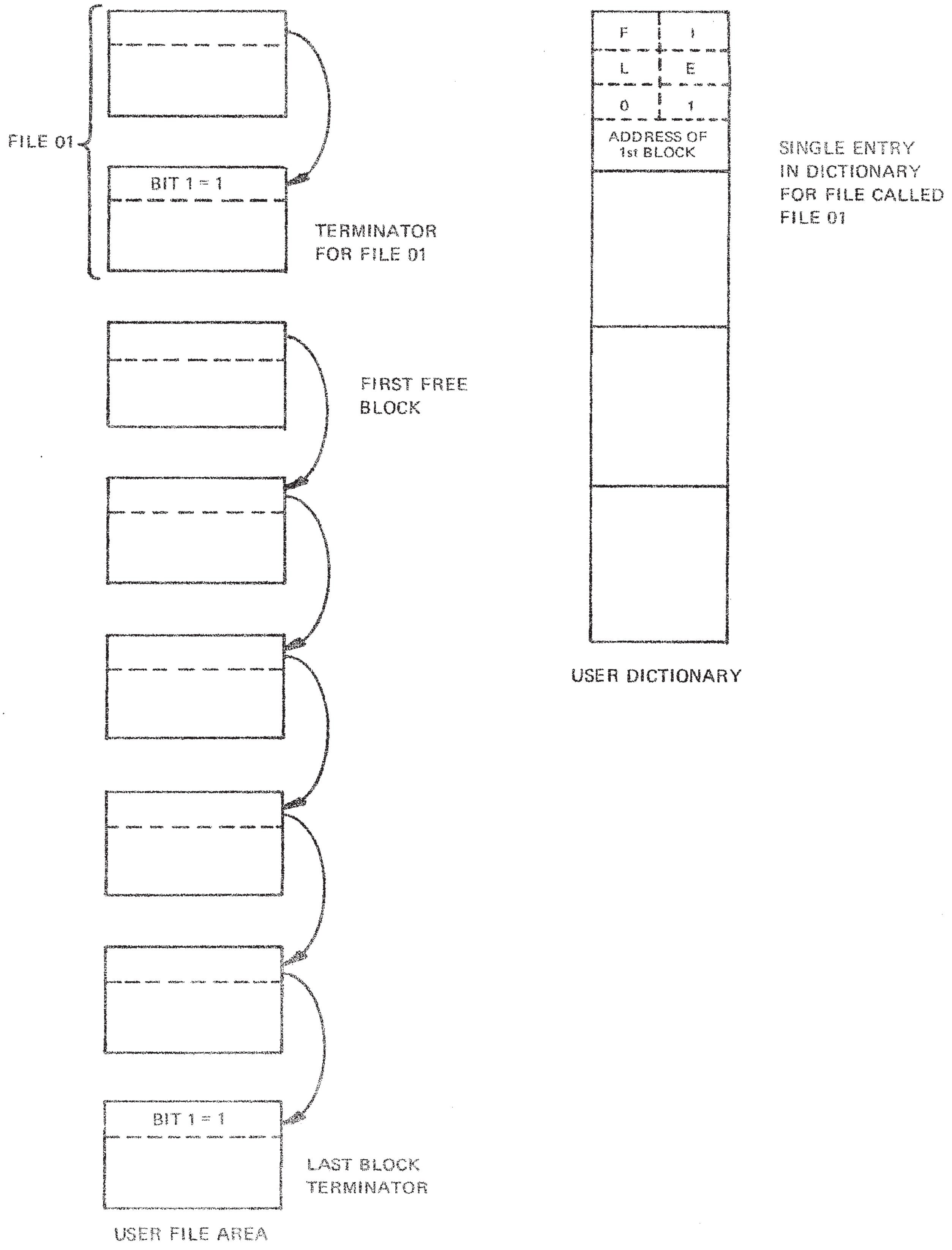
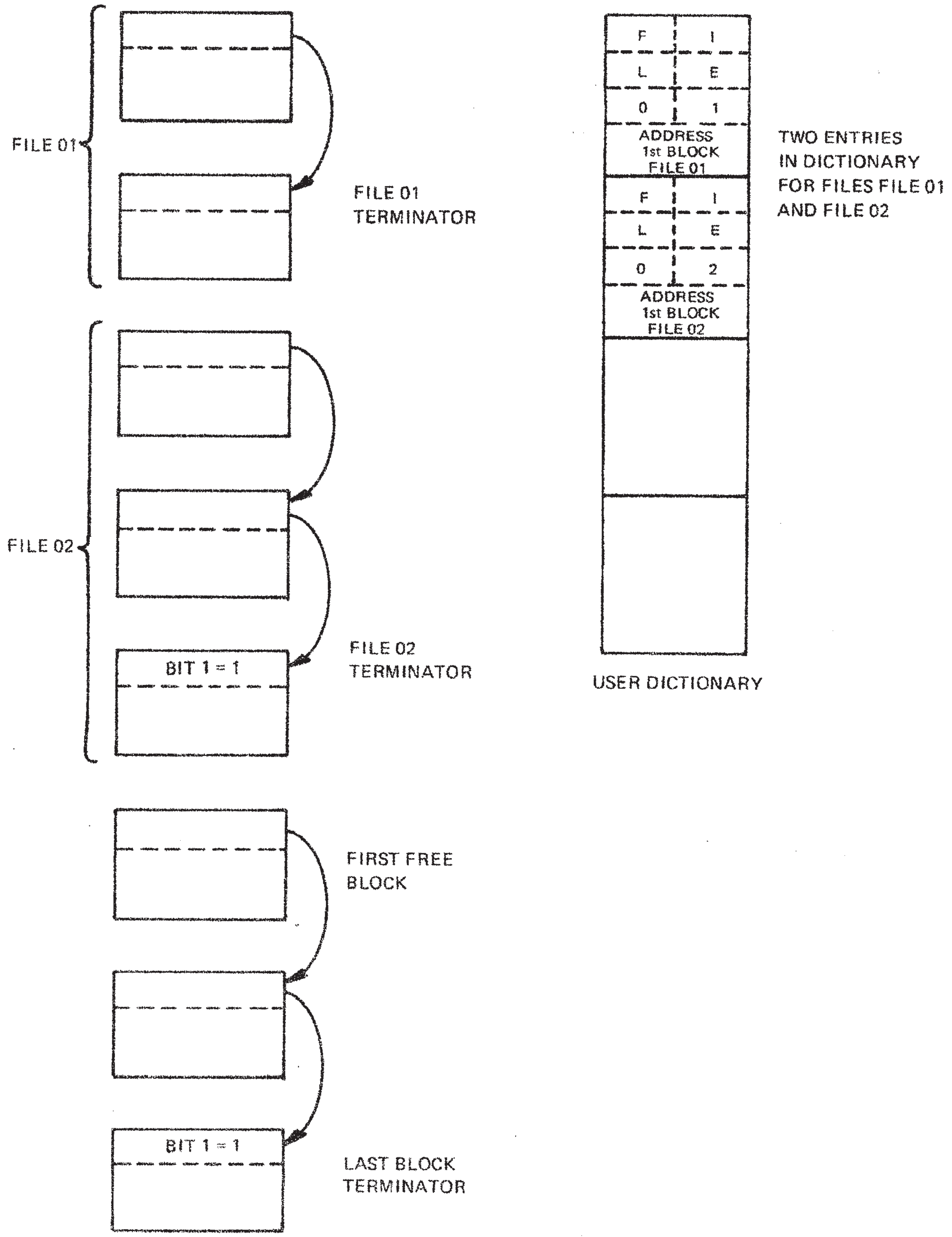


FIG. 8 - 2 USER FILES AFTER ONE FILE IS ESTABLISHED



USER FILES AFTER SECOND FILE ESTABLISHED FIG. 8 - 3

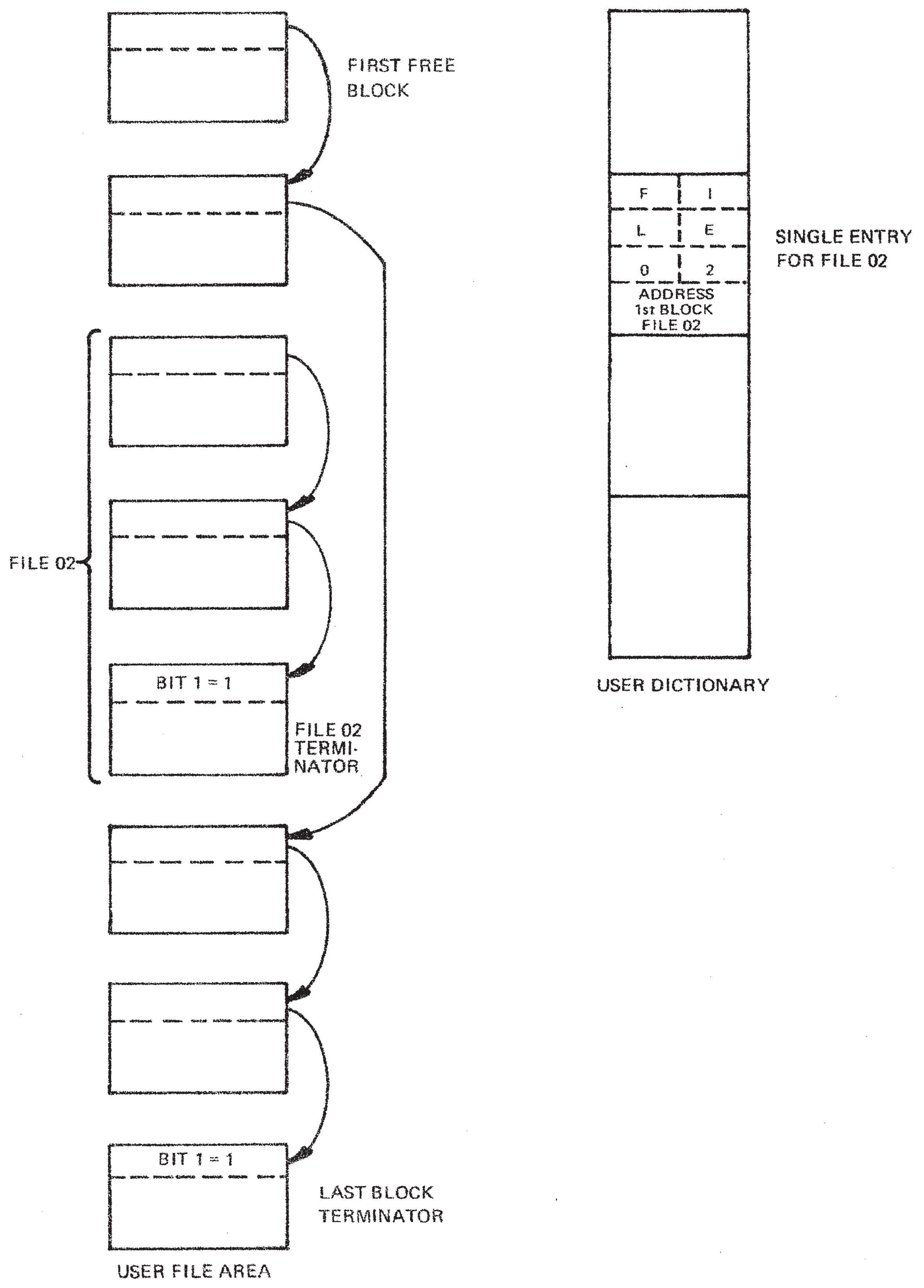


FIG. 8 - 4 USER FILES AFTER FIRST FILE DELETED

(4) The disc scratch area is used for temporary storage of the command file during $DO or $CALL operations. It extends from the block address one higher than the low core common block to the block address one lower than the first block of the User Library.

This area is required to protect a User command file, that may contain a $JO command, from destroying itself

(5) The user library is the area defined by the parameters Q5, Q6 in BSSP (see section 5.2.1). The area is set up in two parts by the $JOB command.

(a) The first block of the user library (the block with the lowest address) is defined as the first block of the user dictionary.

(b) The second block of the user library (the block with the second lowest address) is defined as the first user free block. All other blocks in the user area are linked together in order of increasing block address. The string or chain so formed is known as the user free string.

Descriptions of the format of dictionary entries for each user file, and of each data block in a user file are given in Figs. 8 - 5 and 8 - 6 respectively.

## 8.2.1 Adding a File to the Library

(1) When a file is to be added to the user or system library the action is identical and is described as follows:

The system or user dictionary is searched to see if the file name already exists. If it does the system error ED is generated. If it doesn't the file is written onto the disc in blocks. Every time a new block is required, the first block available on the appropriate free string is allocated to the file. Each block in the file is linked to the next by a pointer word. When the file is complete the last block has its pointer word set –ve to mark this block as the last in the file. A special end-of-file mark will also be written into the last block to mark the end of information within the block. The file name together with the block address of the first block of the file will then be added to the user or system dictionary. Each dictionary will contain (N-3)/5 entries in each block (where N is the number of words per block). When more than this number of names is defined, the system will allocate a new block from either the user or system free string. Figs. 8 - 1 to 8 - 3.

(2) When a file is deleted the system will add the file into the appropriate free string as follows.

The last block of the deleted file has its pointer, normally set –ve to mark the end block, set to the address of the old first free block. The appropriate first free block location in low core common is then set to the address of the first block of the deleted file. Fig. 8 - 4.

## 8.2.2 Contents of File Pointer Table

The BOS file structure enables many files to be accessed at the same time by the use of the file pointer table concept. One file pointer table is required for each file that is currently open. Each file pointer table is N + 9 words long where N is the number of words in each BOS block. The structure of the table is as follows:

| | |
|---|---|
| FPT + 0 | File name characters 1, 2 |
| + 1 | File name characters 3, 4 |

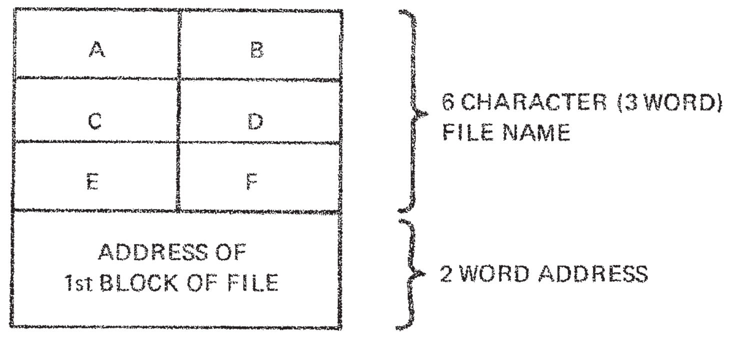


FIG. 8 - 5 FORMAT OF DICTIONARY ENTRY FOR A FILE

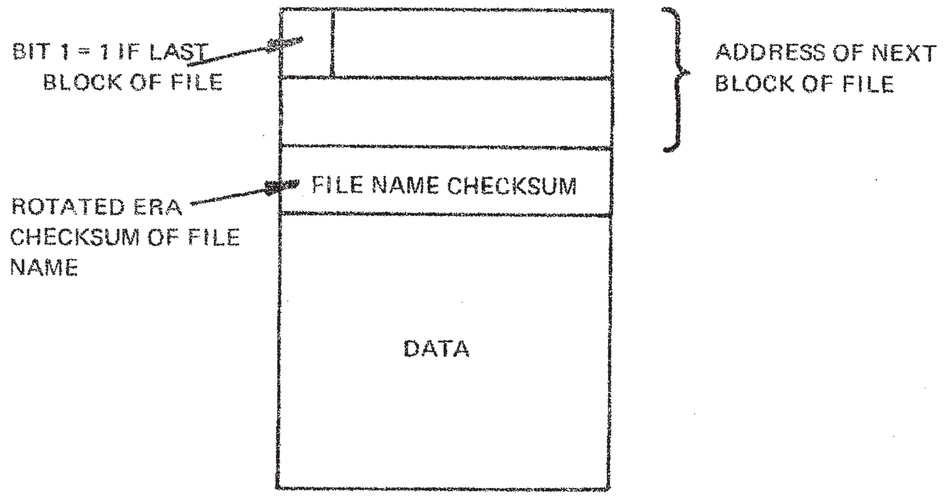

BIT 1 = 1 IF LAST BLOCK OF FILE
ADDRESS OF NEXT BLOCK OF FILE
ROTATED ERA CHECKSUM OF FILE NAME
FILE NAME CHECKSUM
DATA

FIG. 8 - 6 FORMAT OF DATA BLOCK IN USER FILE

| | |
|---|---|
| + 2 | File name characters 5, 6 |
| + 3 | Pointer (next word in data area) |
| + 4 | Counter (negative count of number of words remaining in data area) |
| + 5 | First block of file (most significant half) |
| + 6 | First block of file (least significant half) |
| + 7 | Current block of file (most significant half) |
| + 8 | Current block of file (least significant half) |
| + 9 | Next block of file (most significant half) |
| + 10 | Next block of file (least significant half) |
| + 11 | File name merged |
| + 12 | First word of data |
| + N + 9 | Last word of data |

The area from FPT + 9 to FPT + N + 9 is written onto the disc.

The disc driver initially sets:–

FPT + 3 to DAC FPT + 12

and FPT + 4 to –N

The file name merged is the rotated ERA sum of the 6 character file name and is used as a check of file integrity. The error MF (mixed files) is generated if the file name merged location does not agree in every block of a file.

The most significant bit of the first character of the file name is used to indicate whether the file is reading or writing. If the MS bit is 1 the file is considered to be reading, if it is zero the file is writing.

## 8.2.3 BOS Internal Record Format

Within each file information is recorded in the form of records. Each record consists of a header word followed by L information words.

The format of the header word is '125LLT where LL is the record length in the range 0-60 and T is the record type 0-7

where type 0 is a binary record
1 is an end of file record
2 is an end of group record
3 is an object record
4 is a source record
5 is an error record
6 is a special record
7 is an illegal record type

The record length for types 1, 2 and 7 is always 0. The record type 7 (illegal) is only used to follow a record type 1 (end of file) to ensure that any attempt to read past the end of file record results in a system error RT (illegal record type). The record length for binary records is always 2 and the format of the record is as follows

| | |
|---|---|
| '125020 | header word |
| 'SSSSSS | Binary file size (words) |
| 'FFFFFF | First core address of binary file |

This is followed by 'SSSSSS words consisting of binary information.

If the file was established as a result of a $ES command or a $LO command that resulted in an LC (load complete) the binary record will be followed by an object (type 3) record containing the following 2 words of information:-

(1) Word 2 contains the re-entry point of the file i.e. the first location to be executed following the return from an overlaid program.

(2) Word 1 contains the normal (initial) entry point of the file.

## 8.3 BOS Low Core Common Area

BOS uses an area of core between addresses '22 and '75 for storage of parameters defining the status of the system at the current time. This area is written onto the disc periodically and is restored following a bootstrap restart. The usage of this area is given in Table 8 - 1.

TABLE 8 - 1
LOCATION/USE OF LOW CORE COMMON AREA

| Location | Use |
|---|---|
| '22 | Device Flag. This is -ve if a line printer is required for system messages. |
| '23 | System Mode. This is non-zero if in system mode. Zero if not in system mode. When the BOS main system is in core, BOS is said to be in system mode. This parameter is -ve if the current job is being aborted. |
| '24 | Memory size (e.g. 40000 for 16K) |
| '25 | Number of Parameters. This is the number of parameters present in the command record, not counting the command itself. |
| '26 | Address of First Parameter. This is the address of the first parameter in high core. |
| '27 | Address of Restart Routine (e.g. 37000) for a 16K machine) |
| '30 | Memory Mode (EXA or DXA) |
| '31 | JMP* '27. This instruction is executed to return control to BOS. |
| '32<br>'33 | Address of System Dictionary. This is the block address of the first block of the system dictionary. |
| '34<br>'35 | Address of System First Free Block. This is the block address of the system free string. |
| '36<br>'37 | Address of User Dictionary. This is the block address of the first block of the user dictionary. |
| '40<br>'41 | Address of User First Free Block. This is the block address of the first block of the user free string. |

Continued

Table 8 - 1 / Continued.

| Location | Use |
|---|---|
| '42<br>'43 | Address of User Unused Area. This is the block address of the first block not yet used in the user library. This parameter marks the terminating point of $JOB commands. |
| The Stream Assignment table contains the symbolic names of the devices currently attached to each I/O stream. The streams and their respective locations are as follows: | |
| '44 | Command Input device |
| '45 | Source Input device |
| '46 | Object Input device |
| '47 | Binary Input device |
| '50 | Error Output device |
| '51 | Source Output device |
| '52 | Object Output device |
| '53 | Binary Output device |
| '54 | XAC B$RB Address of the read a block routine |
| '55 | XAC B$WB Address of write a block routine |
| '56 | XAC B$CA Address of convert block address routine |
| '57 | XAC B$ER Address of the Error routine |
| '60<br>'61 | 32 Bit Senselight word |
| '62 | Current position of magnetic tape (file count).<br>This is used by $PO commands. |
| '63 | Used for the start button interrupt link |
| '64 | Required position of magnetic tape (file count)<br>This is the XX parameter of the $PO command |
| '65 | Final position of magnetic tape (file count)<br>This is the YY parameter of the $PO command. |
| '66 | DO Record Address (most significant half) |
| '67 | DO Record Address (least significant half) |
| '70 | Number of words per block |
| '71<br>'72<br>'73 | File name of $DO command file |
| '74 | Record Address of $CALL (most significant half) |
| '75 | Record Address for $CALL (least significant half) |

## 8.4 Return to BOS System Mode (General)

### 8.4.1 Initial Start

Initial start is achieved by using the BOS bootstrap tape. This tape, punched in 8 - 8 format is a special self loading tape of the top sector of the BOS system (B$CO and B$RB). After being read into core a jump is made to the first address of the top sector in core with the A-register equal to the contents of 'A' before the key in loader was entered (this is normally 0).

### 8.4.2 Top Sector Restart

Top Sector Restart is achieved by manually starting BOS at the first location of the top sector of core. The A-register contains the parameter '0' (normally) to define manual restart.

### 8.4.3 End of Subsystem Restart

This type of restart is normally achieved by the DAP-16 or FORTRAN statement:- CALL EXIT. Actual return to BOS within the subroutine EXIT is achieved by the DAP-16 statements:-

```
LDA     SW
JMP     '30
```

Location '30 in the Low Core Common Block contains coding necessary for a return to BOS. The location SW contains a value 0 - 7 to define the type of BOS return. For normal return this is 3.

### 8.4.4 Possible BOS Return Parameters

The parameter in the A-register on entry to the BOS core resident routine B$CO defines the mode of the restart. The Parameter must be in the range 0 - 7. The action is:-

(1) The BOS main system is loaded into core from location '100 to location B$TM.

(2) The sector containing B$EF is loaded into the sector below the restart routine.

Further action depends on the value of the parameter as follows:-

A=0 Manual Restart (Either from a top sector restart or a bootstrap restart). The action is:-

(1) The low core common block is loaded into core.

(2) The message RESTART is output to the ASR.

(3) The user library is set up so that a subsequent $JO command will clear the entire library.

(4) The command input is attached to the ASR.

A=1 Start button interrupt. This occurs following a JST '63 caused by the start button being pressed. The action is:

(1) The Low Core Common block is written to disc.

(2) The message INT is output on the ASR.

(3) The command input is attached to the ASR.

A=2 Unsolicited Key in. This occurs if a key on the ASR is hit during I/O on a device other than the ASR. The action is:-

(1) The Low Core Common block is written to disc.

(2) The message UNS is output on the ASR.

(3) The command input is attached to the ASR.

A=3 Normal return from subsystem -- Low Core Common Preserved, The action is:-

(1) The Low Core Common block is written to disc.

(2) The next command is read on the command input device.

A=4 Normal return from subsystem -- Low Core Common Destroyed. The action is:-

(1) The Low Core Common block is loaded into core from disc.

(2) The next command is read on the command input device.

A=5 System Error. This occurs for any error detected by BOS. The action is:-

(1) The Low Core Common block is written to disc.

(2) The message SE XX is output on the ASR where XX identifies the error.

(3) The system is put into ABORT mode so that only compulsory commands are obeyed.

(4) The next command is read on the command input device.

A=6 FORTRAN Run Time Error. This occurs when a routine calls the FORTRAN error processor F$ER. The action is:-

(1) The Low Core Common block is written to disc.

(2) The message FE XX is output on the ASR where XX identifies the error.

(3) The system is put into ABORT mode.

(4) The next command is read on the command input device.

A=7 User Error. This occurs when a routine calls the User Error procedure. The action is:-

(1) The Low Core Common block is written to disc.

(2) The Error message UE XX is output on the ASR where XX identifies the error.

(3) The system is put into ABORT mode.

(4) The next record is input from the Command Input device.

Note: If A is equal to any value outside the range 0 to 7, the return is made with an assumed System Error IR ( Illegal Return to System Mode).

(3) The command input is attached to the ASR.

A=2 Unsolicited Key in. This occurs if a key on the ASR is hit during I/O on a device other than the ASR. The action is:–

(1) The Low Core Common block is written to disc.

(2) The message UNS is output on the ASR.

(3) The command input is attached to the ASR.

A=3 Normal return from subsystem – Low Core Common Preserved, The action is:–

(1) The Low Core Common block is written to disc.

(2) The next command is read on the command input device.

A=4 Normal return from subsystem – Low Core Common Destroyed. The action is:–

(1) The Low Core Common block is loaded into core from disc.

(2) The next command is read on the command input device.

A=5 System Error. This occurs for any error detected by BOS. The action is:-

(1) The Low Core Common block is written to disc.

(2) The message SE XX is output on the ASR where XX identifies the error.

(3) The system is put into ABORT mode so that only compulsory commands are obeyed.

(4) The next command is read on the command input device.

A=6 FORTRAN Run Time Error. This occurs when a routine calls the FORTRAN error processor F$ER. The action is:–

(1) The Low Core Common block is written to disc.

(2) The message FE XX is output on the ASR where XX identifies the error.

(3) The system is put into ABORT mode.

(4) The next command is read on the command input device.

A=7 User Error. This occurs when a routine calls the User Error procedure. The action is:-

(1) The Low Core Common block is written to disc.

(2) The Error message UE XX is output on the ASR where XX identifies the error.

(3) The system is put into ABORT mode.

(4) The next record is input from the Command Input device.

Note: If A is equal to any value outside the range 0 to 7, the return is made with an assumed System Error IR ( Illegal Return to System Mode).

## 8.5 DAP-16/BOS Interface

This subsection deals with the program interface available to the DAP-16 programmer writing programs to run under BOS. These programs are mostly concerned with device and file handling, but also include error and normal returns to BOS.

### 8.5.1 Device Handling

Devices may be handled by a DAP-16 program in the normal way by making calls on the standard I/O library. The calling sequences for the standard drivers is given in the relevant manuals and listings. However, should the programmer wish to drive the device directly from his program, the following point should be noted.

BOS does not use interrupt control for any I/O. All device interrupts are masked out and should remain so. Since the devices always raise interrupt requests when they operate, there will often be outstanding interrupts from devices after the BOS drivers have used the device. This should not normally cause any concern unless the user wishes to drive a device under interrupt control. In this case he must ensure that before masking in the relevant interrupt, any outstanding interrupt from a previous operation must be acknowledged by giving an appropriate instruction to the device controller (usually an OCP instruction).

### 8.5.2 File Handling – General Rules

The routines which are available on the BOS library tape are as follows:-

| | | |
|---|---|---|
| B$EF | Establish a file (for writing) | |
| B$OF | Open a file (for reading) | |
| B$WR | Write a record to file | |
| B$WW | Write a word to file | |
| B$RR | Read a record from file | |
| B$RW | Read a word from file | |
| B$DF | Delete a file from the backing store | |
| B$SU | Check if system or user file | Note: These routines are |
| B$DS | Search for a file in the dictionary | always used together |

The following rules must be observed when using the above routines.

(1) Each file that is open must be allocated a file pointer table. This table is used by the backing store driver as an I/O buffer. This table is 9 words longer than the BOS block size, i.e. 104 words long for 95 word BOS.

The first 3 words of the table must be set by the user to the name of the file being accessed. The 6 characters name must be packed 2 characters per word and padded with blanks if necessary. Apart from this initial setup, the table must not be modified in any way during the time the file is open.

(2) The transfer of records to disc (as opposed to words) require a record buffer table as the BOS interface. Only one table is required, no matter how many files are open. This table is 63 words long and is arranged as follows:-

| | |
|---|---|
| Word 1 | (RBT) is used for housekeeping |
| Word 2 | (RT) is the record type (0-6) |
| Word 3 | (RL) is the record length (0-60 words) |
| Words 4-63 | (RB) are the record buffer area. |

(3) The X register must contain the address of the relevant file pointer table before each routine is called.

(4) If files are written in word mode the user is responsible for ensuring that the necessary header words for each record are generated to comply with the BOS file standards. (See section 8.2.3. – BOS Internal Record Formats.)

(5) No user program may overwrite any location between '22 and '77. The only exception to this rule is that location '63 (the standard interrupt link) may be modified by the user. Failure to observe this rule will lead to unspecified system malfunction which may not be detected until a later time.

(6) Records read back from the backing store are not padded with blanks (or zeros, in the case of object records). It is the user's responsibility to fill the record buffer with blanks or zeros before reading the record from the backing store.

(7) To conserve backing store space it is normal practice to strip trailing blanks from source records and trailing zeros from object records when writing files. It is the users responsibility to perform this action. (Note: The standard routine C$SW in GEN-IOL may be used.)

## 8.5.3 Calling Sequences for File Handling Facilities

### 8.5.3.1 To establish a file to be written into

```
LDX     AFPT      ADDRESS OF FILE POINTER TABLE
CALL    B$EF      ESTABLISH A FILE
RETURN
```

Possible errors are:

```
SE      DO        No more room on backing store
SE      ED        Attempt to establish a duplicate file
SE      MF        Mixed Files
```

### 8.5.3.2 To open a file for reading

```
LDX     AFPT      ADDRESS OF FILE POINTER TABLE
CALL    B$OF      OPEN A FILE
RETURN
```

Possible errors are:

```
SE      MF        Mixed Files
SE      ON        Attempt to open a non-existent file
```

### 8.5.3.3 To write a record to file

```
LDX     APFT      ADDRESS OF FILE POINTER TABLE
CALL    B$WR      WRITE A RECORD
DAC     RBT       ADDRESS OF RECORD BUFFER TABLE
RETURN
```

Possible errors are:

| | | |
|---|---|---|
| SE | MF | Mixed Files |
| SE | RT | Illegal record type |
| SE | RL | Illegal record length |
| SE | DO | No more room on backing store |

8.5.3.4 To write a word to file

```
LDX      AFPT      ADDRESS OF FILE POINTER TABLE
LDA      WORD      WORD TO BE WRITTEN
CALL     B$WW      WRITE A WORD TO FILE
RETURN
```

Possible errors are:

| | | |
|---|---|---|
| SE | MF | Mixed Files |
| SE | DO | No more room on backing store |

8.5.3.5 To read a record from file

```
LDX      AFPT      ADDRESS OF FILE POINTER TABLE
CALL     B$RR      READ A RECORD FROM FILE
DAC      RBT       ADDRESS OF RECORD BUFFER TABLE
RETURN             BINARY RECORD  (TYPE 0)
RETURN             END OF FILE RECORD (TYPE 1)
RETURN             END OF GROUP RECORD (TYPE 2)
RETURN             OBJECT RECORD (TYPE 3)
RETURN             SOURCE RECORD (TYPE 4)
RETURN             ERROR RECORD (TYPE 5)
RETURN             SPECIAL RECORD (TYPE 6)
```

Possible errors are:

| | | |
|---|---|---|
| SE | MF | Mixed Files |
| SE | RT | Illegal record type (TYPE 7) |
| SE | RL | Illegal record length (>60) |
| SE | RF | Illegal record format (header word error) |

8.5.3.6 To read a word from file

```
LDX      AFPT      ADDRESS OF FILE POINTER TABLE
CALL     B$RW      READ A WORD FROM FILE
RETURN             WORD IN A REG.
```

Possible error is:-

| | | |
|---|---|---|
| SE | MF | Mixed Files |

8.5.3.7 To delete a file

```
        LDX     AFPT        ADDRESS OF FILE POINTER TABLE
        CALL    B$DF        DELETE A FILE
        RETURN
```

Possible errors are

|  |  |  |
|---|---|---|
| SE | DN | Attempt to delete a non-existent file |
| SE | MF | Mixed Files |

8.5.3.8 Search for a file in the dictionary

```
        LDX     AFPT        ADDRESS OF FILE POINTER TABLE
        CALL    B$SU        CHECKS SYSTEM OR USER FILE
        CALL    B$DS        SEARCH DICTIONARY
        RETURN 1            FILE NOT FOUND
        RETURN 2            FILE FOUND
```

Possible error is:

|  |  |  |
|---|---|---|
| SE | MF | Mixed Files |

8.5.4 Use of System File Pointer Table

To conserve space the system file pointer table may be used by user programs. This table is normally located immediately below B$FP. The address of this table is stored within B$CO and may be obtained by the following sequence.

```
        LDX     '27         ADDRESS OF RESTART ROUTINE B$CO
        LDA     6,1         FETCH ADDRESS OF FILE POINTER TABLE
        STA     AFPT        ADDRESS OF FILE POINTER TABLE
```

8.5.5 Normal Return to BOS

Normal return to BOS can be achieved either by the statement

```
        CALL    EXIT
```

or by the sequence

```
        LDA     =3          NORMAL RETURN PARAMETER
        JMP     '30         RETURN TO BOS
```

This will cause Low Core Common to be written back to the disc.

The sequence.

```
        LDA     =4
        JMP     '30
```

will cause Low Core Common to be restored from the disc. This must be used if low core common has been corrupted. (Note:- Locations '27, '30 and '31 must never be overwritten if return is to be made to BOS).

## 8.5.6 Error Returns to BOS

8.5.6.1 *System Error Return* – System error returns to BOS can be achieved by the DAP-16 statements:-

```
        JST*    '57     CALL ERROR ROUTINE
        BCI     1,XX    ERROR IDENTIFIER
```

This causes a return to BOS in ABORT mode with the message

```
        SE      XX      printed on the ASR
```

8.5.6.2 *User Error Return* – A user error return may be implemented by the sequence

```
        CALL    ERROR   ERROR RETURN
        DAC     =AXX    ERROR IDENTIFIER
```

or by the sequence

```
        LDA     =7      USER ERROR PARAMETER
        LDX     =AXX    ERROR IDENTIFIER TO X
        JMP     '30     RETURN TO BOS
```

This causes a return to BOS in ABORT mode with the message

```
        UE      XX      printed on the ASR
```

## 8.5.7 Use of System Mass Store Driver

The system mass store driver may be used to read or write any BOS type block from the mass store. The user must specify the block address of the block to be read or written in the file pointer table locations, FPT+7 and FPT+8.

The calling sequence for using the system mass store is:

```
    LDX     AFPT    ADDRESS OF FILE POINTER TABLE
    JST*    '54     READ A BLOCK
    OR
    LDX     AFPT    ADDRESS OF FILE POINTER TABLE
    JST*    '55     WRITE A BLOCK
```

WARNING: Writing blocks in the BOS system area may cause the system to be corrupted. These routines should only be used for the area of disc outside BOS executive control.

# 9 USE OF COMMAND FILES

## 9.1 Introduction to Command Files

From time to time it is found that the same sequences of commands are being used repeatedly in different jobs. An example of this might be updating, compiling, and output of listing and object tape. It would be very convenient if these common sequences of operations could be saved on the disc, and invoked as required. This can be done by setting up a 'Command File' on the disc.

A Command File is basically a file of source records. Each record is a valid BOS command. To enable a command file to be established, the '$' in column one must be changed to a '%' sign. When BOS executes the commands from the file the '%' is changed back to a '$'.

Three basic commands are provided to facilitate the use of command files. These are:

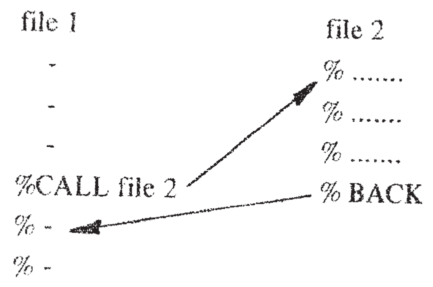
```
DO      <file name>
CALL    <file name>
BACK
```

When the DO command is executed either from a normal command input stream (e.g. ASR, CR) or another command file, the command input stream becomes attached to the file named in the command. BOS then continues to execute commands from that file. To return control to another stream, the appropriate

```
%AT   CI,    <device>
```

must be included in the command file.

Although control may be passed from one command file to another by use of the DO command, it is often useful to be able to call a command file as a 'subroutine', and be able to return to the calling command file at the next command in sequence. The CALL and BACK commands enable this to be done. When the %CALL <file 2> command is executed from command 'file 1', command input will be transferred to 'file 2'. When a %BACK command is executed from 'file 2', command input will return to the next sequential command in file 1 following the %CALL command.



Two other commands are included to enable the writing of more complex command files. These are:

```
%BDISC    n
%FDISC    n
```

These enable commands to be skipped backwards or forwards within a command file.

A further facility is provided to enable command input to be attached to the disc directly. The use of this is when a command file is being executed, as the result of a

```
$DO    <file name>
```

command input from the ASR. The command file then returns control to the ASR by executing a

```
%AT     CI, AK
```

The operator may now type in further commands and then resume the execution of the command file by typing

```
$AT     CI,DI
```

This can only be used to return to a command file.

Another use of this facility is to set up source files from within a command file. e.g.

```
%AT     SI, DI
%ES     F1, SI
REC 1
REC 2
%AT     CI, AK
```

This sequence will establish a file called 'F1' containing 2 records. Control is then returned to the ASR.

## 9.2 Rules for Constructing Command Files

The following points should be noted when constructing BOS command files.

(1) All records within the command file must not be more than 22 (decimal) characters long. Any character in excess of 22 will be ignored.

(2) The maximum size of a command file is defined at system configuration time. The BOS configurator configures BOS to allow a maximum command file of 384 records. If a $DO or $CA file overflows the available space in the scratch area, a system error FL is generated.

(3) A command file executed as a result of a $DO command must have either a %AT CI, XX command as the last command executed (where XX is the name of a command input device), or a %DO YY as the last command executed to begin obeying commands in file YY.

(4) A command file executed by a %CA command can either:-

(a) Terminate as for a %DO file as defined in section 3.

(b) Terminate on a %BA command if control is to be returned to the command following the %CA.

(5) %CA commands cannot be nested, i.e. a file executing as a result of a %CA command may not contain any further %CA command.

(6) %BA commands are illegal unless encountered in a file executing as a result of a %CA command.

(7) Care must be taken when constructing command files to avoid the establishing of duplicate files. The %NO command may be used to test if a file with a given name exists, e.g. the sequence:

```
%NO FRED
%DE FRED
```

deletes a file called FRED if it exists.

Caution: The above sequence will fail if FRED exists but is empty.

(8) Command files should arrange to trap any system errors that may occur in BOS subsystems, e.g. to trap DAP-16 source errors (system error SE).

```
%DA  S,O,L
%FD  4
%RE
%TY
JOB ABORTED
%AT CI, AK
/
JOB CONTINUES
```

The above sequence will proceed normally if the assembly produces no system errors but will perform a controlled abort if a system error occurs.

(9) A record not starting on a % will be treated as a comment and ignored, unless immediately preceded by a %TY or %NE command.

(10) The %FD and %BD commands refer to records within a file, not just to command records, hence comments and text lines must be counted (in octal) when calculating the %FD or %BD parameters.

(11) When establishing source files from text within a command file, the source input must be attached to the disc, i.e. the sequence:

```
%AT  SI, DI
%SI  LOD
LO
```

will establish a file called LOD containing a single record consisting of the text LO.

## 9.3 Examples of BOS Command Files

### 9.3.1 Example 1 – Compile, load, and Execute a FORTRAN Program

Note: A non-standard subsystem $FO is used in this to compile FORTRAN source file(s), into an object file (O), producing a listing file (L).

```
BOS JOB FILE F4C

%AT SI,PR
%ES F4C,SI
%TY
F4C COMPILES SOURCE
%AT SI,PR
%NO S                   DELETE ANY OLD SOURCE OBJECT OR LISTING
%DE S
%NO O
%DE O
%NO L
%DE L
%WA
```

```
WAIT FOR USER TO LOAD TAPE AND HIT CR
%ES S,SI
%FO S,O,L              COMPILE THE PROGRAM
%AT SO,LP
%OU L,SO               LIST THE LISTING FILE
%ER L                  TEST FOR ERRORS
%AT CI,AK              ABORT IF ERRORS
%NO M                  DELETE ANY MAP OR BINARY FILES
%DE M
%NO R
%DE R
%LO O,$SL,*M=M,*R=R    LOAD THE PROGRAM
%OU M,SO               PRINT THE MAP
%EX R                  EXECUTE USERS PROGRAM
%RE
%AT CI,AK              END OF JOB
```

## 9.3.2 Example 2 – DAP-16 Assembly

This job makes use of two $DO files. The second one is called by the first using a %CALL command. The job establishes and assembles three DAP-16 source files. The actual establishing and assembly is performed by the second $DO file. This file outputs listings on the line printer, and also checks for errors in the assembly and aborts the job if there are any. If not, it returns to the first $DO file using a %BACK command. The first $DO file concludes by loading the object files and outputting a loader map on the line printer. SENSLIGHT 1 is used to request loading as well as assembly.

```
 BOS JOB FILE $AL3

$AT SI,PR
$ES $AL3,SI
%AT SI,DI
%NO ESF                TEST IF NAMES ARE ALREADY SET
%FD 14                 THEY ARE, SKIP THE SET UP
%SI ESF                SET UP COMMAND NAMES
ES
%SI SIF
SI
%SI NOF
NO
%SI DEF
DE
%SI DAF
DA
%SI LF
L
%NO S                  DELETE FILE S IF IT EXISTS
%DE S
%SI S                  SET UP FILE 1 NAMES
S1
%SI O
O1
%CA $AL3X1             ESTABLISH/ASSEMBLE SOURCE FILE 1
%SI S                  SET UP FILE 2 NAMES
S2
%SI O
O2
```

```
%CA $AL3X1              ESTABLISH/ASSEMBLE SOURCE FILE 2
%SI S                   SET UP FILE 3 NAMES
S3
%SI O
O3
%CA $AL3X1              ESTABLISH/ASSEMBLE SOURCE FILE 3
%IF 41                  TEST IF LOADING REQUESTED
%FD 7                   NOT, SKIP OVER LOAD
%NG OB=O1,O2,O3         FORM 1 OBJECT FILE FOR LOADING
%NO M                   DELETE OLD MAP AND RESULT FILES (IF THEY EXIST)
%DE M
%NO R
%DE R
%LO OB,$SL,*M=M,*R=R
%OU M,SO                PRINT A MAP
%AT SI,PR               RESTORE DEVICE FOR END OF JOB
%TY
END OF JOB
%RE
%AT CI,AK
$END
```

```
BOS COMMAND MODULE $AL3X1
$ES $AL3X1,SI
%CO NOF,S               DELETE SOURCE IF IT EXISTS
%CO DEF,S
%CO NOF,O               DELETE OLD OBJECT IF IT EXISTS
%CO DEF,O
%TY
LOAD SOURCE IN PR
%AT SI,PR
%WA
%CO ESF,S,SIF           VARIABLE ESTABLISH COMMAND
%RE
%AT SI,DI
%CO DAF,S,O,LF          VARIABLE DAP-16 ASSEMBLY COMMAND
%FD 2                   FORWARD IF NO SOURCE ERRORS
%RE
%AT CI,AK               ABORT IF SOURCE ERRORS
%ER L                   TEST FOR ASSEMBLY ERRORS
%FD 4                   PROCESS ASSEMBLY ERRORS
%AT SO,LP               PRINT LISTING ON LINE PRINTER
%OU L,SO
%DE L,S,O               DELETE UNWANTED FILES
%BA
%TY
ERRORS
%AT EO,AP               OUTPUT ERROR LINES
%OU L,EO
%SE O,O                 INHIBIT LOADING
%BD 10                  CONTINUE
$END
$RE
$AT CI,AK
```

# 10 SORT PROGRAM

## 10.1 Description

BOS provides a multi-key disc facility for sorting files of source records. The characteristics of the SORT program are:

(1) Up to 14 sort keys may be specified. Each key is defined by a pair of octal numbers giving the first and last columns occupied by the key field. If a single column key field is to be defined, both numbers must be the same, and equal to the required column.

(2) Records within the file to be sorted (i.e. the input file) may be variable in length, with a maximum length of 120 characters.

(3) The size of the file that may be sorted is restricted only by the amount of backing store available. To operate successfully, the SORT program requires a minimum area of;

2X number of blocks occupied by the input file + 3 blocks.

This space is in addition to the space occupied by the input file itself.

(4) The collating sequence is defined by the ISO value of the character (See Appendix C). e.g. a space ('240) is lower than an A ('301).

(5) The sort may be executed in either ascending or descending order of character values, the values being those defined by the collating sequence.

(6) A restart facility is provided to enable the merge phase of the process to be restarted without the need to re-execute the sort. This is useful if a system failure should occur during the merge operation.

(7) Random files must be made sequential before they can be sorted.

## 10.2 Use of the Sort Program

SORT is provided as a self establishing subsystem called $SRT. This is loaded into the BOS system in the same way as the other subsystems (see Section 5.4). Once established, it may be used by executing the following command:

$EX $SRT, <file 1>, <file 2>, A or D, S or R, $K_1, K_2$, $K_3, K_4, \ldots, K_n, K_{n+1}, \ldots,$

where :

| | | |
|---|---|---|
| $SRT | – | name of subsystem |
| file 1 | – | name of file to be sorted (input file) |
| file 2 | – | name to be given to sorted file (output file) |
| A or D | – | A for sort in ascending order<br>D for sort in descending order |
| S or R | – | S for normal sort<br>R for restart of merge pass |
| $K_1, K_2$<br>$K_3, K_4$<br>,<br>. . .<br>,<br>$K_n, K_{n+1}$<br>,<br>. . .<br>, | – | pairs of octal numbers defining the key fields to be sorted. Up to 14 fields may be specified. When more than one field is specified the priority of the keys is from left to right – the left most key having the highest priority i.e. $(K_7, K_8)$ is sorted within $(K_5,K_6)$ within $(K_3,K_4)$ within $(K_1,K_2)$. |

Example:

We wish to sort a file of source records called FILE1, in ascending order, each record having the following format:

| | | |
|---|---|---|
| 1257642198 | ABXZ1253YPQX | A |
| field 1<br>numeric<br>(10 digits<br>columns 1 to 10) | field 2<br>alphanumeric<br>(12 characters<br>columns 17 to 28) | field 3<br>alphanumeric<br>(1 character<br>column 39) |

The file is to be sorted primarily on the single character field 3, and then it is to be sorted on field 2 within field 1. i.e. the priority of the keys is field 3, and then it is to be sorted on field 2 within field 1. i.e. the priority of the keys is field 3, field 1, field 2. The output file is to be called FILE2

The required command is:

```
$EX $SRT,FILE1,FILE2,A,S,47,47,1,12,21,34
```

(Note: these keys are specified by octal numbers).

The result of this sort (FILE2) might look like this:

```
1123451169      AAAB1111XYZP     A
1123451169      AAAC1111XYZP     A
1123451170      AAAC1111XYZP     A
1123451058      AAAD1111XYZQ     B
1123451058      AADD1111XYZQ     B
1123451059      AAAA1111XYZQ     B
```

## 10.3 Note on Restart Facility and Errors

During the sort process, should a system error occur, the process may be restarted by typing in a command identical to the original command, but with Restart specified instead of Sort (parameter 5 of $EX command). If the process was in the merge phase, this will be restarted. Should the error have occurred in the sort phase, a system error 'SE ON' will occur on attempting to restart. The whole operation must then be restarted by re-typing the original command. At the end of the merge, if the output file name specified is not unique, an 'SE ED' error will occur. This may be remedied by inputting a restart command with a different output file name.

# 11 EXTERNAL RECORD FORMATS

The operating system works essentially in ISO code, also known as CCITT 5. The coding details for all the available characters are given in Appendix C, the Series 16 Peripheral Device Codes table. The internal code used is such that it always has a 'one' in the channel eight position (bit 9).

## 11.1 Console Keyboard (ASR)

When the operating system requires input from the console keyboard (ASR), it will output a question mark (?) at the start of the next line. Either even-parity or forced-parity records may be input, but no check is made for the correct parity. A source record may consist of any number of printable characters and spaces and is followed by a carriage return (CR, '015 or '215). After the 120th character, any subsequent characters occurring in the record are ignored. If required, a line can be deleted by typing a commercial 'at' sign (@, '300). The left arrow character (←, '137 or '377) causes input to be backspaced one column. A backspace should not immediately follow a backslash or control-tab character, neither should a backslash or control-tab immediately follow a backspace.

Tab stops are set up in columns 6, 12, 30 and 73. A control tab (HT, '011, '211 or control I) or a backslash (\, '134 or '334) will produce spaces up to and excluding the next tab stop. An end-of-group record consists of any source record containing an ETX character (Control C, '003 or '203). An end-of-file record must be a valid BOS command or simply $END. Non-printing characters, other than those mentioned above, are ignored.

## 11.2 Paper Tape Reader

This section gives details of the standard formats for source and object modes of operation.

### 11.2.1 Source Mode

In source mode, paper tape is read in under control of the standard library routine I$PA, initialised to read 120-character records with tab stops in columns 6, 12, 30 and 73. Either even-parity or forced-eight parity records may be input. The type of parity checking to be performed is selected when the system is configured. The standard format for a source line is as follows:

1. *Line feed (LF, '012 or '212).*
2. *Any number of printing characters and spaces.*
3. *Carriage return (CR, '215).*
4. *$DC_3$ ('223, control S).*
5. *DEL ('377, rubout).*

However, if the tape is to be read by the high-speed paper tape reader only, it may be punched as *2*, *3* and *1* only, in that order.

An end-of-group record consists of the following characters:

1. *ETX ('003 or '203).*
2. *$DC_3$ ('223).*

3. *DEL ('377).*

Again, if the tape is to be read by the paper tape reader only, items *2* and *3* may be omitted. An end-of-file record consists of any source record with a dollar sign ($, '044 or '244) in column 1. An end-of-file record must be a valid BOS command or simple $END.

### 11.2.2 Object Mode

The standard format for an object record is:

1. *SOH ('201).*
2. *A multiple of three data characters in invisible format.*
3. *$DC_3$ ('223).*
4. *DEL ('377).*

The DEL may be omitted if the tape will be read only by the high-speed paper tape reader. An end-of-group record may either be as described for source or be an object record with block type 0-2.

An end-of-file record consists of an object record of block type 0-3. This consists of the following four words in invisible format: '000300, '000000. '000005, '000305.

## 11.3 Paper Tape Punch

This section gives details of the standard formats for source and object modes of operation.

### 11.3.1 Source Mode

Each file that is output to the paper tape punch in source mode is preceded by a length of blank tape (NUL '000) and followed by a length of blank tape which is then followed by an end-of-file record (LF, $END, CR, $DC_3$, DEL) and a further length of blank tape.

Each source or error record is output as a string of ISO code characters, using tab stops as described for Source Input (Section 11.2). Each record is preceded by LF ('212) and followed by CR ('215), $DC_3$ ('223) and DEL ('377). Output is always in the parity as selected at configuration time. Each end-of-group record is punched as ETX ('003 or '203), $DC_3$, ('223) and DEL ('377) followed by a length of blank tape.

### 11.3.2 Object Mode

Each file that is output to the paper tape punch in object mode is preceded by a length of blank tape (NUL, '000) and followed by a length of blank tape which is then followed by an object end-of-file record and a further length of blank tape. Each object record is output as a multiple of three characters in invisible format and is preceded by SOH ('201) and followed by $DC_3$ ('223) and DEL ('377). Each end-of-group record is punched as ETX ('003 or '203), $DC_3$ ('223) and DEL ('377), followed by a length of blank tape.

## 11.4 Card Reader

This section gives details of the standard formats for source and object modes of operation.

### 11.4.1 Source Mode

Source cards should be punched in one of the Series 16 Card Codes.

On input, characters are converted from card code by use of a conversion table. The conversion table is selected from those provided in GEN-IOL when the system library is built.

An end-of-group record consists of any card having an 11-8-6 punch in one or more columns; any other information present on the card is ignored. It is conventional to punch an ETX (11-8-6) in column 1 and leave the rest of the card blank.

An end-of-file record consists of any source record containing a dollar sign ($) in column 1. This record must be a valid BOS command or simply $END. It should be noted that if EBCDIC code is in use, a semi-colon (;) character gives an 11-8-6 punch.

### 11.4.2 Object Mode

Object cards should be punched in 80 column binary. The first word is read from column 1, rows 12 to 9 and column 2, rows 12 to 1. The second word is read from column 2, rows 2 to 9 and column 3, rows 12 to 5. The third word is read from column 3, rows 6 to 9 and column 4, rows 12 to 9. This sequence is repeated to obtain words 4 to 60 from columns 5 to 80. There is no room for the punching of sequence numbers since all 80 columns are needed to represent an object record of 60 words.

An end-of-file record is punched as a column binary card as described under Object Mode and the following four octal words: '000300, '000000, '000005, '000305 followed by zero's up to the end of the card.

## 11.5 Card Punch

This section gives details of the standard formats for source and object modes.

### 11.5.1 Source Mode

80 character records are punched using all 80 columns of the card. Code conversion from internal ISO code to the required card code is performed according to a translation table selected when the system library is configured. The available conversion tables are located in GEN-IOL.

An end-of-group record is punched as a card having an ETX (11-8-6) punch in column 1. The rest of the card is left blank. It should be noted that if EBCDIC code is in use, an end-of-group record gives a semi-colon (11-8-6) punch and so should be avoided in text to be output.

An end-of-file record is punched as a card containing the character $END in columns 1 to 4. The rest of the card is left blank.

### 11.5.2 Object Mode

Cards are punched according to the standards described for the card reader -- object mode (Section 11.4.2).

## 11.6 Magnetic Tape

This section gives details of the standard formats for source and object modes.

### 11.6.1 Source Mode

Source tapes should be recorded using the new Series 16 standard 6-bit code as shown in the 'SIX-BIT CODE' column of Appendix C in BCD in even-parity mode. BOS executes the necessary code conversions, that is, from 6-bit code to ISO code on input and from ISO code to 6-bit code on output. All the ISO code characters shown in Appendix C are available with the exception of backslash (\) and left arrow (←) since these are used on paper tape to represent 'tab' and 'delete' respectively.

The 6-bit code '00 is converted and written onto magnetic tapes as '12; when this is subsequently read back, the hardware reconverts it to '00 again. The 6-bit codes '12 and '56 are not employed since the latter is the equivalent of ETX (11-8-6) on cards.

Blocking is not used and each block on the tape contains one, and only one, record. The input routine attempts to read 120 character records; shorter records than this are filled with spaces on the right. For longer records, the excess over 120 characters is ignored. The output routine invariably writes 120 character records.

There is no representation on magnetic tape of end-of-group records in a file on the disc. During the execution of an output command, these end-of-group records are ignored and could cause a VERIFY error (SE VG) if the output was subsequently re-ESTABLISHed and verified.

End-of-file is represented by the presence of a tape mark. During the execution of an ESTABLISH command, a source record starting with a dollar sign ($, '53) is also interpreted as an end-of-file record. An end-of-file record must be a valid BOS command or simply $END. It is not possible to OUTPUT, under BOS control, records having a dollar sign in the first character position, except by using $NEXT command.

### 11.6.2 Object Mode

Object tapes should be recorded in binary in odd-parity mode using three characters per word. The first character is stored in bits 1 to 6 of the word, the second character in bits 7 to 12, and the least significant four bits of the third character in bits 13 to 16. Records may contain from 1 to 60 words.

An object end-of-group is recorded as an object record containing the following four octal words: '000200, '000000, '000005, '000205. An object end-of-file is represented by a tape mark or by an object record containing the following four octal words: '000300, '000000, '000005, '000305.

# 12 DEBUG AND TRACE PROGRAM

## 12.1 Description

The BOS Debug and Trace Program provides facilities for modifying displaying and tracing BOS binary files and enables the transfer and manipulation of binary files between core and disc.

The program may be used either in the conversational mode (command input via the ASR keyboard), or in the remote mode (command input from a BOS source file). Output is available via the ASR or line printer.

The program is available for all sizes of BOS except 8K. The subsystem together with the BOS core resident routine, file handling routines (B$EF) and file pointer tables occupy the top '14000 words of core, eg for 16K BOS, the subsystem uses all core above its start address of '24000. All the program to be debugged must reside in core below the subsystem start address. Also, the program being debugged should be executed in BOS addressing mode.

The source input normally consists of two letters defining a command, followed by a list of zero or more parameters each separated by a comma, and terminated by a semi-colon. A command may occupy more than one record, and more than one command may occupy a record. Each record consists of not more than 120 ISO-code characters followed by a carriage return. The space character ('240) is generally ignored (except in headings or comments), and may be used freely for formatting.

Parameters, which are defined by the first character of the parameter (e.g. + or — for decimal, or 0–7 for octal), may be general (e.g. value), or specific (e.g. address, must be octal). For any parameter used the appropriate conversion routine must be included in the system.

Address parameters always consist of 1 to 5 octal digits, and may be followed by a + sign, the effect of which is to add a relocation factor (previously set), to the address before internal use. All address parameters are checked for violation of the area of core occupied by the Trace and Debug Subsystem and the object computer core size.

Comments are identified by an asterisk in the first column of a record, and may be inserted anywhere. The whole record, which is treated as a comment, is normally output to the source output device (unless the ASR is being used for both input and output), and is then ignored.

If the ASR is being used for source input then the program will request further input by outputting 'CRLF: '.

A source line may be deleted by typing a commercial at (@, '300).

The source output consists of a string of characters not exceeding 120 per record. The individual format of the string of characters depends on the modules used, but usually consists of address parameters and value parameters.

Address parameters consist of 5 octal digits, and the relocation factor is subtracted before output. If the address is less than the re-location factor, it is output as an absolute address (5 octal digits) followed by the letter A character.

Value parameters are output in the current output mode, which has previously been set.

Sense switch four may be set to terminate those commands which may produce a large amount of output (e.g. display core). Normally operation is resumed when sense switch four is reset.

## 12.2 Control Commands

The notations used in the description of commands is as follows:

| | |
|---|---|
| A | Address parameter (1 to 5 octal digits) |
| P | Parameter, may be in any format for which a conversion routine is provided, and the binary value does not exceed 16 bits |
| M | Octal Mask (1 to 6 octal digits) |
| F | Octal Numerical Parameter (1 to 6 octal digits) |
| B | BOS Filename |

SO; Set Output Mode to Octal. The effect of this command is for all subsequent parameters, output in the current output mode, to be output as 6-digit octal numbers.

SR A; Set Relocation. The relocation factor is set to the value of the address parameter A.

MC A, P1, ... , PN; Modify Core. The parameters P1 to PN are stored sequentially in the locations starting at address A.

The number of locations modified is limited only by an attempt to overwrite the Debug and Trace Subsystem. The command is obeyed up to the delimiter (semi-colon), or the detection of an address or parameter error.

FC A1, A2, P, M; Fill Core. The locations from address A1 to address A2 are filled with the value of parameter P. Only those bits corresponding to the optional mask are filled.

If the mask is omitted a default value of '177777 is used.

DC A1, A2, F; Display Core. The contents of locations, address A1 to address A2, are displayed in the current output mode, at F locations per record.

Each record displayed is preceded by the address of the first location of the record. The value F may be omitted, in which case it is given a default value (normally one). The address A2 and value F may be omitted, in which case only the contents of A1 are displayed. If the values are the same throughout a record, and equal to the last value output, then the record is not output, unless it is the last record. A blank record is displayed to indicate when this has happened.

It is the users responsibility to ensure that the value of F chosen will not cause the record length to exceed the capacity of the output device. If sense switch four is set the command is terminated.

SE A1, A2, P, M; Search Equal. The contents of locations, address A1 to address A2, are compared with the value of parameter P for equality.

Only those bits corresponding to the optional mask are compared.

If the mask is omitted a default value of '177777 is used.

If a location is found equal, the address of the location is output, followed by the value in the current output mode.

If sense switch four is set the search is terminated.

SU A1, A2, P, M; Search Unequal. This is identical to the 'Search Equal' command (section preceeding this), except that the search (and corresponding output) are for inequality.

CC A1, A2, A3, M; Copy Core. The contents of locations, address A1 to address A2, are copied sequentially into the area of core starting with address A3. Only those bits corresponding to the optional mask are copied.

If the mask is omitted a default value of '177777 is used.

N.B. No check is made on whether the two areas of core overlay each other.

VC A1, A2, A3, M; Verify Core. The contents of locations, address A1 to address A2 are verified sequentially with the area of core starting with address A3.

Only those bits corresponding to the optional mask are verified.

If the mask is omitted a default value of '177777 is used.

Failure of two locations to verify causes their addresses and values to be output. Values are output in the current output mode.

If sense switch four is set the verification is terminated.

SA P; Set User A-register. The user A-register is set to the value of parameter P.

SB P; Set User B-register. The user B-register is set to the value of parameter P.

SX P; Set User X-register. The user X-register is set to the value of parameter P.

SK F; Set User Keys. The user keys are set to the octal value of parameter F.

DR; Display Registers. The user registers (including keys) are displayed.

The format is preceded by the character R ('322), which is followed by the registers A, B, and X, followed by the keys.

The registers are displayed in the current output mode, and the keys in octal.

BP A; Set Breakpoint. A breakpoint is set at location A. If the user program is subsequently executed by the 'START USER PROGRAM' Command (see section ST A;) and reaches location A, then control is returned to the Debug and Trace Subsystem. When this occurs the following record is output:

```
B A P
```

where A is the address of the breakpoint and P is the contents in the current output mode. The breakpoint remains active (i.e. it is set up each time the 'START USER PROGRAM' command is used) until it is cleared.

BP; Clear Breakpoint. This clears the breakpoint if previously set.

ST A; Start User Program. The user registers and keys are set up, and the user program is entered at location A. If a breakpoint has been set, then the return link is inserted before entering the user program, in the mode of BOS.

The user program must run in the same mode (extended or normal) as BOS, otherwise control may be lost.

ST; Continue User Program. The user program is continued from the last breakpoint. For this to be effective the breakpoint must be moved before this command is obeyed. If this command is used before a breakpoint has been hit, then the user program is continued from '1000.

TR A1, A2, F1, F2, A3, A4: This is the main Trace start command and must be used each time it is desired to enter the Trace routine. In order that the module may be restarted as quickly as possible, all seven parameters need not be specified, but they can be eliminated from the right, and those parameters which have not been re-specified retain their previous values. All the parameters have default values (see below). The parameter list is terminated at any stage by a semi-colon — any parameters specified following a semi-colon will not be read.

The meaning of the various parameters with their default values is given below:

A1 - (default '1000) - Trace start address - the address within the user's program at which tracing will commence.

A2 (default '30000) - Upper limit for Trace. On reaching this address, Trace ceases and a new command is requested.

F1 . (default 2) - Trace mode. This parameter has three possible values:

0 - No Trace output. May be used if a run is required solely for timing purposes.

1 - Jump Trace. Only JMP and JST statements are printed, together with a compulsory register print-out, with the values contained therein after the jump has been effected.

2 - Full Trace. All instructions printed according to the format described below.

F2 - (default 1) - Number of instructions to be traced, divided by '100. A count is kept of the number of instructions traced, and when this exceeds the specified value, the error message NE is output. F2 is limited to 2 octal digits, so that the maximum number of consecutive instructions which can be traced is 4032 decimal.

A3 - (default '1000) - Lower limit of user's program.

A4 - (default '30000) - Upper limit of user's program.

These two parameters are used in checks to see that various parts of the system do not overwrite each other.

There is one other entry point to Trace, which sets up areas of core in which output from the Trace program is inhibited. The command format is:

```
TA S1,E1,S2,E2,...S6,E6;
```

The parameters S and E are start and end addresses of areas in which output is inhibited. A typical application of the use of this facility would be for the start and end of a subroutine which is called several times in the course of a segment of program being traced, and which is known to be bug-free. The amount of output to be examined is then considerably decreased. The output inhibit flag thus set over-rides all other print options.

On entry to the TA section, the output inhibit buffer is zeroised and the inhibit flag set off. Thus the 1st, 3rd, 5th . . . parameters specify addresses at which inhibition starts, and the 2nd, 4th, 6th . . . those at which it ceases.

The format of the output of the Trace package is as follows:

| | |
|---|---|
| Field 1 | (P) 5 octal digits - The address within the user program of the instruction currently being traced. |
| Field 2 | (INS) 6 octal digits - The contents of the address specified in field 1,ie.the user program instruction as it is stored within the computer. |
| Field 3 | (OP ADF) 3 letter mnemonic, followed by qualifier dependent upon the type of instruction - The DAP-16 mnemonic corresponding to the current instruction, with an indication given of indirect addressing and indexing along with the address, number of shifts, etc. or merely blank, if there are no qualifiers here. |

Fields 4 and 5 apply only to memory reference instructions.

| | |
|---|---|
| Field 4 | (MDAD) 5 octal digits - An address in the chain of indirect addressing. |
| Field 5 | (CONT) 6 octal digits - The contents of the address in field 4. |
| | If indirect addressing is used, the contents themselves are addresses until the chain is terminated. Thus for each level of indirect addressing, there are entries in fields 4 and 5. If the instruction decoded is in double precision, the two memory locations involved, and their contents are printed successively in fields 4 and 5. |
| Fields 6 - 9 | (A,B,X,C) 6,6,6,1 octal digits - The contents of the registers after execution of the instruction just traced. The value of a register is only printed if its value has changed since the last time it was printed. This rule is not necessarily followed in the case of jump instructions (JMP or JST), where all the registers are always printed. |

WARNING: If Trace is used on coding which requires response to interrupts,eg I/O instructions dealing with card readers, etc., the method which the program uses will cause the timing to go wrong.

AT SO, YY; Attach Source Output. Source output may be attached to the ASR printer (AP) or line printer (LP).

SPX...Y; Set Page Heading. The character string X to Y is output to the line-printer source heading buffer, where it is output subsequently at the top of each page used, starting at page one.

The character string may not exceed 120 characters and may be further restricted by the line-printer driver. Space characters ('240) are not ignored in the string.

SD; Set Output Mode to Decimal. This causes all subsequent parameters, output in the current output mode, to be output as decimal numbers.

The output format is 6 characters, consisting of a sign (space character or --), followed by 1 to 5 decimal digits, leading zeros are suppressed and the sign is right justified.

EXAMPLES: 16000 -9 7081

SI; Set Output Mode to ISO-code. This causes all subsequent parameters, output in the current output mode, to be output as two ISO-code characters.

The output format is two ISO-code characters (in range '240 to '337). If either one or both characters are outside the range, then two space characters are output.

EXAMPLES: AB Z3 ↑

SH; Set Output Mode to Half-Words. This causes all subsequent parameters, output in the current output mode, to be output as half-words.

The output format is two three-digit octal numbers separated by a comma.

EXAMPLES: 317,120 010,000 177,100

SS; Set Output Mode to Symbolic. This causes all subsequent parameters, output in the current output mode, to be output either as a DAP-16 Symbolic instruction if valid, or as an octal parameter if invalid.

The output format is a DAP-16 mnemonic and then for each instruction type, as follows:

Memory Reference: An asterisk if indirect, space if not, followed by colon if indexed, space if not, followed by an address parameter. If the double precision mode is set in the User's Keys, then affected instructions will be output in that mode.

EXAMPLES: LDA 01250 STA* 0010 DLD: 15600

Input/Output: Two space characters followed by four octal digits.

EXAMPLES: INA 0004 SMK 0020 OTA 0204

Shift: Two spaces followed by a plus sign character followed by 1 or 2 decimal digits in range 0 to 32.

EXAMPLES: LGR +4 LRR +29 LRL +32

Generic: No parameters

EXAMPLES HLT INK SS4

SF; Set Output Mode to Full Format. This causes all subsequent parameters, output in the current output mode, to be output in octal, half-words, decimal, ISO-code and symbolic formats.

| EXAMPLES: | 000201 | 000,201 | 129 | IAB |
|---|---|---|---|---|
| | 140702 | 301,302 | -15934 AB | 140702 |
| | 140320 | 300,320 | -16176 @P | CSA |

RC B, A1, A2; Transfer Result File from Disc to Core. The binary file B is brought into core between the address limits A1 and A2. The entry point is printed, if one exists.

RD B,(A1,A2),(A3,A4),....,($A_{n-2}$,$A_{n-1}$),$A_n$; Transfer Result File from Core to Disc. Core locations A1 through A2,A3 through A4, up to $A_{n-2}$ through $A_{n-1}$ are written to disc file B together with the entry point $A_n$ if specified. At least one pair of addresses (ie A1,A2) must be specified.

EB; Enter BOS. The user is returned to the BOS command mode.

## 12.3 Input Parameters

The following types of parameters may be input (each parameter must be terminated by a comma or semi-colon):

### 12.3.1 Decimal Parameters

+ or – sign followed by 1 to 5 decimal digits

eg +39, –15879 +2;

### 12.3.2 ISO-Code Parameters

Quotation mark followed by two ISO-code characters (in range '240 to '336), followed by quotation mark.

eg "A2" "2"; "␣↑",

### 12.3.3 Half-Word Parameters

< sign followed by 1 to 3 octal digits (first half word), comma followed by 1 to 3 octal digits (second half word) and > sign.

eg <216,0>; <377,2>; <1,1>,

### 12.3.4 Symbolic Parameters

Modified DAP-16 format for each type of instruction as follows:

Memory Reference - DAP-16 mnemonic, an asterisk, if indirect, colon if indexed, and an address parameter followed by a plus sign if relocated.

eg STA 50+; LDA* 1050, IRS : 3760

Input/Output - DAP-16 mnemonic followed by one to four octal digits.

eg OCP4; SKS 0002, OTA 204

Shift - DAP-16 mnemonic, a plus sign followed by 1 or 2 decimal digits in the range 0 to 32.

| | | | |
|---|---|---|---|
| eg | ALS +0; | ARR +9; | LLS +32; |

Generic - DAP-16 mnemonic

| | | | |
|---|---|---|---|
| eg | IAB; | NOP; | SRZ; |

## 12.4 Error Conditions

All commands are checked for the following error conditions set out below. On finding an error the command is terminated (ie all characters up to, and including the next semi-colon, are ignored), and the appropriate error mnemonic is sent to the source output device.

### 12.4.1 Illegal Command

This error is caused by the command mnemonic not being recognised or the command not being terminated by a semi-colon character if no parameters are required.

Error message DE IC

### 12.4.2 Illegal Parameter

This error is caused by either a parameter not being recognised, or the parameter contains an illegal character (eg decimal digit in an octal parameter).

Error message DE IP

### 12.4.3 Parameter Missing

This error is caused by a command being terminated before it has received the minimum number of parameters.

Error message DE PM

### 12.4.4 Extra Parameter

This error is caused by a command not being terminated after it has received the maximum number of parameters.

Error message DE EP

### 12.4.5 Illegal Address Parameter

This error is caused by an address parameter being generated that either exceeds the object computer core limit, or violates the core area occupied by the Debug and Trace Subsystem.

Error message DE IA

## 12.5 Execution

The debug program is executed by the command

```
$EX $DT, <file>
```

where <file> is a source file containing the debug commands or,

```
$EX $DT
```

In this case debug command input is attached to the ASR keyboard. Command input is requested by the characters colon space (: ) being output.

## 12.6 Examples

The following examples illustrate two uses of debug:

(1) This example illustrates the use of debug with the commands held in a file DCFILE.

DCFILE contains

```
AT SO, AP;
RC TEST, 100,1777;
* BASE SECTOR OF TEST
DC 100,777; 10;
* PROGRAM AREA OF TEST
SS; DC1000, 1777;
EB;
$END
```

TEST is a binary file containing a program that uses the first 3 sectors of memory

```
? $EX $DT,DCFILE                  BOS command entered by user
EP = 1000
* BASE SECTOR OF TEST
00100-001700-001706 . . .       }
00110                           }
  .                             }
  .                             }  Base sector display
  .                             }
00770                           }
* PROGRAM AREA OF TEST
01000 LDA*  00100               }
01001 STA*  00101               }
  .                             }  Program area display
  .                             }
  .                             }
01777 HLT                       }
?
```

The program held in file TEST is brought into memory. Its base sector is displayed in octal format and program area displayed in symbolic format. Return is then made to BOS command mode.

(2) This example illustrates the use of debug in conversational mode.

```
? $EX $DT
:  AT  SO,AP;
:  RC  TEST,100,1777;
EP = 1000
:  SS;  DC 1020;
01020 SZE
:  MC  1020, SNZ;
:  RD TEST1,100,1777,1000;
:  EB;
?
```

The program held in file TEST is brought into core. A location is then displayed and subsequently modified. The resulting program is then written to a file called TEST1, and return is made to BOS command mode.

# APPENDIX A

# ERROR MESSAGES

When a system error occurs, a message of the form 'SE pq' is output on the ASR. For installations possessing a line printer, a further message is also printed in the form:

SYSTEM ERROR pq – JOB ABORTED

In the case of the latter, the message is printed out on a page by itself. In both instances the characters pq represent one of the two letter mnemonics from the table of System Errors (see Table A-1). The system error marker is set to inhibit the execution of most commands. Brief details of system errors are given in Table A-1. Further details, if required, may be obtained from the appropriate listing. It is possible that system error codes not included in this list may appear. This is because it is possible for a user program or subsystem to request a system error by making the appropriate return to BOS. The I/O library also generates some system errors which are not listed here.

**TABLE A-1**
**SYSTEM ERRORS**

| System Error | Explanation |
|---|---|
| AB | Abort command executed |
| BA | Nested back commands |
| BD | % BDisc command out of control |
| BF | Binary file incorrectly formatted |
| BL | Loader block error |
| BO | Bootstrap tape error |
| BR | Binary record found |
| BS | Illegal disc block size |
| CA | Nested call commands |
| CK | Checksum error |
| CW | Illegal command word |
| DA | Disc access error or illegal address |
| DC | DO file flows out of control |
| DN | Attempt to delete a non-existent file |
| DO | Disc Overflow |
| DP | Disc off-line or write protect |
| DT | Disc data transfer error |
| DV | Disc verification error |
| ED | Attempt to establish a duplicate file |
| ER | End of mag. tape reached while reading |
| EW | End of mag.tape reached while writing |
| FD | %FDisc command out of control |
| FL | File too large for scratch area |
| IC | Illegal character in command |
| IM | Writing to a file in input mode |
| IP | Incorrect parameter in command |

TABLE A-1 (Continued)

| System Error | Explanation |
|---|---|
| IR | Incorrect return to system mode |
| LO | Loader parameter list error |
| MF | Mixed file |
| MO | Memory overflow |
| NA | Device not available |
| NO | FORTRAN program attempted to access an unopened random file |
| NP | Incorrect number of parameters in command |
| NR | FORTRAN program attempted to open a non-random file for random access |
| OL | Character other than letter in command |
| OM | Reading from a file in output mode |
| ON | Attempt to open a non-existent file |
| OT | FORTRAN program attempted to open a random file twice |
| RF | Illegal record format |
| RL | Illegal record length |
| RN | FORTRAN program attempted to access a record outside the last record in a random file |
| RT | Illegal record type |
| RU | Mag.Tape record unreadable |
| SD | Incompatible stream and device |
| SE | DAP-16 source error (No END pseudo-op) |
| SN | Incorrect stream name |
| TM | FORTRAN program attempted to have open more than 5 random files simultaneously |
| UE | Updating error |
| VA | Verification error (addresses differ) |
| VE | Verification error (record contents differ) |
| VF | Verification error (file lengths differ) |
| VG | Verification error (group lengths differ) |
| VL | Verification error (record lengths differ) |
| VT | Verification error (record types differ) |

# APPENDIX B

# SUBSYSTEM AND LIBRARY TAPE DESCRIPTIONS

This Appendix gives a brief description of the available BOS subsystems. Although complete at the time of going to press, it is not necessarily a complete list at any given time since new subsystems are always under development by Honeywell Information Systems to add to the facilities available under BOS. A full list of available subsystems is included in the Binder Table of Contents for BOS (EBTCBOS3) and the most recent revision of this document can be obtained from Honeywell Information Systems.

## 1 Subsystems

Section 5 describes the format of self-establishing subsystems tapes. The preconfigured tape list for each subsystem (which carries the same name and document number as the subsystem tape itself) gives details of the contents and the way in which the particular tape is configured.

(1) $AN DAP-16 Annotate

This is a subsystem of BOS which makes it possible, within a BOS environment, to annotate an existing DAP-16 Mod 2 source file or old master and produce a new file called the new master. Note that only two files are involved.

(2) $DA DAP-16 Mod 2 Assembler

This is a subsystem of BOS which makes it possible within a BOS environment to do two-pass DAP-16 Mod 2 assemblies of source files stored on the disc, to produce object and listing files on the disc.

(3) $DE Debug Package

This is a subsystem of BOS which makes it possible, within a BOS environment to debug a program that has just been loaded.

(4) $DT Debug and Trace Subsystem

The Debug and Trace Subsystem provides facilities for modifying, displaying and tracing BOS binary files and enables transfer and manipulation of binary files between core and disc.

(5) $FN FORTRAN Translator

This subsystem accepts FORTRAN source and translates this into DAP-16 Mod 2 source. The two files (FORTRAN input and DAP-16 Mod 2 output) are stored on the disc together with a listing file. The DAP-16 Mod 2 source file may be assembled using the $DA subsystem.

(6) $FO FORTRAN compiler

This is a subsystem of BOS which makes it possible, within a BOS environment, to do one-pass FORTRAN compilations of source files stored on the disc, to produce object and listing files on the disc. Note that the object code is restricted to the LXD mode.

(7) $LO Relocating Loader

This is a subsystem of BOS which makes it possible, within a BOS environment, to load into core absolute or relocatable object programs stored on the disc backing store.

(8) $ED Source Edit Program

This BOS subsystem enables a file to be edited in either batch mode (commands held in a command file) or conversational mode (commands typed in from ASR).

(9) $SRT Multi-key Disc Sort

This is a subsystem of BOS which enables a source file to be sortedup to a maximum of 13 pairs of sort keys. The records may be variable in length with a maximum length of 120 characters. The size of file that can be sorted depends upon the backing store space available. (Refer to Section 10 for details).

(10) $TI Disc Tidy Program

This is a subsystem of BOS which enables free space in the user area of the disc to be re-linked without destroying existing user files. It also checks the integrity of files, and has limited capability of overcoming faults on the disc tracks by chaining round bad patches on the disc.

(11) $DI Dictionary

This is a subsystem of BOS which enables a dictionary file to be produced which contains information of all system and/or user files on the disc. The information given includes the file name, file type and file size. The dictionary file can be output on a source output stream to obtain the dictionary information for reference.

(12) $BO Binary Output

This is a special subsystem of BOS which is used by BOS main system output function to output binary files. It is not directly accessible to the BOS user.

## 2 BOS Library Tapes

The detailed contents of the library tapes available can be found in the relevant library tape contents listing which bears the same name and number as the corresponding tape.

(1) BOS-IOL

This tape contains all the various subroutines that are specific to BOS, such things as the FORTRAN file support subroutines – BOS, file handling, etc.

(2) B$MSYS

This tape contains all the mandatory object modules of the BOS main system that can be loaded together initially (see Section 5.2.2).

# APPENDIX C

## SERIES 16 PERIPHERAL DEVICE CODES

| Character | ISO Code | Even Parity | Six-Bit Code | Card Punching | Description |
|---|---|---|---|---|---|
| | 240 | 240 | 20 | blank | space |
| ! | 241 | 041 | 16 | 8-6 | exclamation mark |
| " | 242 | 042 | 37 | 0-8-7 | quotation mark |
| £ | 243 | 243 | 32 | 0-8-2 | currency symbol, pound |
| $ | 244 | 044 | 53 | 11-8-3 | currency symbol, dollar |
| % | 245 | 245 | 75 | 12-8-5 | percent |
| & | 246 | 246 | 77 | 12-8-7 | ampersand |
| ' | 247 | 047 | 14 | 8-4 | apostrophe |
| ( | 250 | 050 | 34 | 0-8-4 | left parenthesis |
| ) | 251 | 251 | 74 | 12-8-4 | right parenthesis |
| * | 252 | 252 | 54 | 11-8-4 | asterisk |
| + | 253 | 053 | 60 | 12 | plus sign |
| , | 254 | 254 | 33 | 0-8-3 | comma |
| - | 255 | 055 | 40 | 11 | hyphen, minus sign |
| . | 256 | 056 | 73 | 12-8-3 | full stop, period, point |
| / | 257 | 257 | 21 | 0-1 | solidus |
| 0 | 260 | 060 | 00 | 0 | zero |
| 1 | 261 | 261 | 01 | 1 | one |
| 2 | 262 | 262 | 02 | 2 | |
| 3 | 263 | 063 | 03 | 3 | |
| 4 | 264 | 264 | 04 | 4 | |
| 5 | 265 | 065 | 05 | 5 | |
| 6 | 266 | 066 | 06 | 6 | |
| 7 | 267 | 267 | 07 | 7 | |
| 8 | 270 | 270 | 10 | 8 | |
| 9 | 271 | 071 | 11 | 9 | |
| : | 272 | 072 | 15 | 8-5 | colon |
| ; | 273 | 273 | 52 | 11-8-2 | semi-colon |
| < | 274 | 074 | 57 | 11-8-7 | less than |
| = | 275 | 275 | 13 | 8-3 | equals |
| > | 276 | 276 | 17 | 8-7 | greater than |

| Character | ISO Code | Even Parity | Six-Bit Code | Card Punching | Description |
|---|---|---|---|---|---|
| ? | 277 | 077 | 35 | 0-8-5 | question mark |
| @ | 300 | 300 | 76 | 12-8-6 | commercial at |
| A | 301 | 101 | 61 | 12-1 | |
| B | 302 | 102 | 62 | 12-2 | |
| C | 303 | 303 | 63 | 12-3 | |
| D | 304 | 104 | 64 | 12-4 | |
| E | 305 | 305 | 65 | 12-5 | |
| F | 306 | 306 | 66 | 12-6 | |
| G | 307 | 107 | 67 | 12-7 | |
| H | 310 | 110 | 70 | 12-8 | |
| I | 311 | 311 | 71 | 12-9 | letter I |
| J | 312 | 312 | 41 | 11-1 | |
| K | 313 | 113 | 42 | 11-2 | |
| L | 314 | 314 | 43 | 11-3 | |
| M | 315 | 115 | 44 | 11-4 | |
| N | 316 | 116 | 45 | 11-5 | |
| O | 317 | 317 | 46 | 11-6 | Letter O |
| P | 320 | 120 | 47 | 11-7 | |
| Q | 321 | 321 | 50 | 11-8 | |
| R | 322 | 322 | 51 | 11-9 | |
| S | 323 | 123 | 22 | 0-2 | |
| T | 324 | 324 | 23 | 0-3 | |
| U | 325 | 125 | 24 | 0-4 | |
| V | 326 | 126 | 25 | 0-5 | |
| W | 327 | 327 | 26 | 0-6 | |
| X | 330 | 330 | 27 | 0-7 | |
| Y | 331 | 131 | 30 | 0-8 | |
| Z | 332 | 132 | 31 | 0-9 | |
| [ | 333 | 333 | 35 | 11-8-5 | left square bracket |
| \ | 334 | 134 | — | — | backslash |
| ] | 335 | 335 | 36 | 0-8-6 | right square bracket |
| ↑ | 336 | 336 | 72 | 12-8-2 | upwards arrow |
| ← | 337 | 137 | — | — | left arrow |

NOTES:

1. On most teletypes £ is printed as #.

2. On some Honeywell line printers, some of the graphic characters may differ from those above.

3. On paper tape, left arrow signifies 'backspace', and backslash means 'tab'; these characters are not available on cards or magnetic tape.

4. The six-bit codes are used for both cards (Hollerith mode) and seven-track magnetic tape (even parity or BCD mode). On writing magnetic tapes, '00 is written as '12; conversely, when reading in even parity, '12 is read as '00. In punched cards, code '56 corresponds to card punching 11-8-6, which is interpreted as ETX (end of text).

5. This particular card code is known as Series 16 Peripheral Device Code. It is possible for a BOS system to be constructed so as to support other card codes; GEN-IOL contains card code conversion tables which support Series 16 Peripheral Device Code, H200 EDP-compatible Code, and a subset of EBCDIC.

# APPENDIX D

## BOS COMMAND SUMMARY & LOADER PARAMETERS

TABLE D-1
BOS COMMAND SUMMARY

| Command | Meaning |
|---|---|
| $AB | Abort the current job |
| $AN OM, NM | Annotate comments from the file named (OM) producing a new file named (NM) |
| $AT SN, DN | Attach the named stream (SN) to the named device (DN) |
| $BA | Return to the next command following the last $CALL |
| $BD 1234 | Move the command pointer backwards 1234 Octal Records in the command file |
| $CA FN | Execute commands in the file called (FN). (The last command in the file (FN) will normally be $BA.) The file called (FN) may not contain any $CA commands. |
| $CO FN1, FN2 etc. | Construct and execute a single command from the Files (FN1), (FN2), etc...... |
| $DA SO, OB, LS | Assemble a source file (SO) using DAP to produce an Object File (OB) and a listing file (LS) |
| $DE FN1, FN2, FN3 etc. | Delete the files named (FN1), (FN2), FN3) etc...... |
| $DI FN | Establish a file called (FN) containing a dictionary of all programs on the disc. |
| $DO FN | Execute commands in the file called (FN) |
| $EN | Used to mark the end of a source file |
| $ER FN | If the file called FN contains any error records obey the next command. If not skip the next command. |
| $ES FN, SN, 1234 | Establish a file named (FN) and read data into the file from the stream named SN. If SN=BI, 1234 is the octal value of the entry point into the program. For SI or OI, 1234 is not required. |
| $EX FN, LIST | Execute the binary file named (FN) with parameters set into high core as defined by (LIST) |
| $FD 1234 | Move the command pointer forwards 1234 octal records in the command file |
| $FI FN1=FN2, FN3 etc. | Construct a file called (FN1) containing the first record only of files (FN2), (FN3) etc. |

TABLE D-1 (Continued)

| Command | Meaning |
|---|---|
| $FN S1, S2, L, OP, C | Translate the FORTRAN source file (S1) into DAP-16 Mod 2, writing the DAP-16 Mod 2 source into file (S2) and the listing into file (L). (OP) is an octal constant specifying the optional compilation, trace, and source output minimisation options required as follows:<br>0 — TRACE incorporated as specified in source; optional statements ignored<br>2 — TRACE incorporated as specified in source; optional statements compiled<br>4 — all statements traced; optional statements ignored<br>6 — all statements traced; optional statements compiled<br>10<br>12 as above for 0, 2, 4, 6 but source output<br>14 minimisation option will be used.<br>16 |
| $FO SO, OB, LS, OP<br><br>*(See Note at the end of Table)* | Compile the FORTRAN source file (SO) to produce an object file (OB) and a listing file (LS). (OP) is an optional octal constant specifying one of the following:<br>0 — symbolic listing<br>10 — expanded symbolic listing<br>5 — trace<br>1 — trace + symbolic listing<br>11 — trace + expanded symbolic listing |
| $IF 23 | Obey the next command if senselight '23 is true, skip the next command if senselight '23 is false. The inverse status of the senselight is tested if '40 is added to the light number (i.e. '63 in this case). Senselights must be in the range 1-30 |
| $JO NAME | Clear up the disc and start the job named (NAME) |
| $KE 2 | Obey the next command if sense-key 2 is true, skip the next command if sense-key 2 is false. The inverse status of the sense-keys are tested if '40 is added to the key number (i.e. '42 in this case). Only keys 2, 3 and 4 can be tested. |
| $LO N1, N2, N3 etc.... | Load object files according to the parameter list N1, N2 etc.... (See Table D-2 for summary of loader parameters) |
| $MA FN1=FN2, FN3 etc. | Make a source or object file named (FN1) out of the files named (FN2) (FN3) etc... |
| $MB DN, 1234 | Backspace the device named (DN) by the specified number of files (Octal 1234) (Mag. tape Only) |
| $MC S1, S2, S3 | For later use with the Macro Subsystem |

[illegible]

TABLE D-1 (Continued)

| Command | Meaning |
|---|---|
| $MF DN,1234 | Forward space the device named (DN) by the specified number of files (Octal 1234) (Mag. Tape Only) |
| $MR DN | Rewind the device named (DN) (Mag. Tape Only) |
| $NA FN1=FN2 | Change the name of file (FN1) to (FN2)..(FN1) and (FN2) must both be either system or user files. |
| $NE | Copy the next record from the command input stream onto the the source output stream. |
| $NG FN1=FN2, FN3 etc... | Make a source or object file named (FN1) out of the files named (FN2), (FN3) etc.. Omit any end of group marks that may be in (FN2), (FN3), etc.... |
| $NO FN | Skip the next command if file (FN) is empty or does not exist. Otherwise obey the next command. |
| $NT | Leave command trace mode |
| $OL FN, SO | Output file (FN) on source output stream and append a line number to each record. |
| $OU FN, SN | Output the contents of the file named FN to the stream named SN. |
| $PO Mn | Position magnetic tape unit to the file specified in the next record to be read from the source input stream. The format of the input stream record is either<br>XX position at file XX (Decimal).<br>XX, YY position at file XX (Decimal) Subsequent $PO commands cause the file position indicator to be incremented by 1 without reading an input record. This continues until XX exceeds YY<br>If XX is zero the tape is rewound and the next command is skipped. |
| $RE | Reset the system error marker |
| $RU | Produce leader (blank tape) on the high speed punch |
| $SE 1234, 5671 | 'AND' the senselight word with 1234 and then exclusive 'OR' the senselight word with 5671. Both octal parameters must be in the range 0-77777777 |
| $SF OM,NM | Delete trailing spaces from all records in file OM to create file NM. |
| $SH OM, NM | Truncate records in the file named OM to produce a new file (NM) with a maximum record length of 72 characters. |
| $SI FN | Read a single record from the source input stream and create a file called (FN) containing it. |
| $TI | Tidy disc — re-links all free blocks in user area, checks integrity of user files and checks faulty blocks. |

TABLE D-2
SUMMARY OF LOADER PARAMETERS

| Parameter | Meaning |
|---|---|
| <filename> | Load the named object file |
| *ADDRESS = n | Set initial address |
| *BASE = n | Set base address |
| *COMMON = n | Set COMMON base address |
| *DEBUG | Input and enter debugging program |
| *EXECUTE | Execute program just loaded |
| *FORCE | Force-load next object module |
| *GTNS | Go to next sector |
| *INITIALISE | Re-initialise loader |
| *LIBRARY | Enter library load mode |
| *MAP = <filename> | Store core-map in named file |
| *NORMAL | Leave extended desectoring mode |
| *OPERATOR | Enter operator (ASR) mode |
| *PBRK = n | Set program break |
| *QUIT | Recall BOS main system |
| *RESULT = <filename>, <optional octal parameter(s)> | Store binary result in named file |
| *START = n | Set start (entry) address |
| *TOTAL | Enter total load mode |
| *VIRTUAL | Enter virtual memory load mode |
| *WORKFILE=<filename> | Enter WORKFILE mode |
| *XTENDED | Enter extended desectoring mode |

**NOTE:** where a value 'n' is specified for an address it must be given as an octal integer.

TABLE D-3
SUMMARY OF EDIT COMMANDS

| Format | Abbreviation | Function |
|---|---|---|
| COPY | C | Insert current line above current line |
| COPY i,j,k | | Copy line i to line j preceding line k |
| COPY i,j,* | | Copy line i to line j preceding the end-of-file |
| COPY ,j,k | | Copy current line to line j preceding line k |
| DELETE | D | Delete current line |
| DELETE i | | Delete line i |
| DELETE i,j | | Delete lines i through j |
| DELETE ,j | | Delete current line through j |
| DELETE i,* | | Delete lines i through end-of-file |
| DELETE ,* | | Delete current line through end-of-file |
| FIND "string" | F | Find next line commencing with string |
| FIND i | | Find the line number i |
| INSERT | I | Insert before current line |
| INSERT i | | Insert before line i |
| †INSERT * | | Insert at end-of-file |
| LOCATE "string 1" | L | Locate the next line with string 1 |
| LOCATE "string 1", i | | Locate next i occurrence of string 1 |
| LOCATE "string 1" ,* | | Locate all following lines occurring with string 1 |
| LOCATE "string1...string 2", i | | Locate next i lines where string 1 is followed by string 2 (separated by any number of characters) |
| LOCATE "string 1", i,j,k | | Locate next i occurrences of string 1 between and including character positions j and k in each line |
| LOCATE "string 1" , j | | Locate the next line with string 1 after and including character position j |
| LOCATE "string 1",,,k | | Locate the next line with string 1 before and including character position k |

The INSERT command is terminated by a record of only an exclamation mark (!) in Column 1 following the data to be inserted.

TABLE D-3 (Continued)

| Format | Abbreviation | Function |
|---|---|---|
| NEXT | N | Move to next line |
| NEXT i | | Move forward i lines |
| NEXT * | | Move to end-of-file |
| NEXT - i | | Move backward i lines |
| PRINT | P | Print current line up to 120 characters |
| PRINT, j | | Print current line up to j characters |
| PRINT i | | Print i lines from current line |
| PRINT i,j | | Print i lines from current line, not more than j characters per line |
| PRINT * | | Print from current lint to end-of-file |
| PRINT *,j | | Print from current line to end-of-file, not more than j characters per line |
| QUIT | Q | Create new master and terminate EDIT |
| READ file 1, file 2 | R | Insert file 1 and file 2 before current line |
| READ n1,file 1,n2,file2 | | Insert file 1 before line n1, and Insert file 2 before line n2 |
| READ *,file 1, file 2 | | Insert file 1 and file 2 at end-of-file |
| SUBSTITUTE "string 1"string 2" | S | Change the next occurrence of string 1 to string 2 |
| SUBSTITUTE "string 1"string 2", i | | Change the next i occurrences of string 1 to string 2 |
| SUBSTITUTE "string 1"string 2", * | | Change all following occurrences of string 1 to string2 |
| SUBSTITUTE "string 1"string 2", i,j,k | | String 2 is substituted for string 1 on i lines where string 1 falls between and includes character positions j and k |
| SUBSTITUTE "string 1"string 2", $i_1$, $j_1$,$k_1$,"string 3"string 4", $i_2$,$j_2$,$k_2$, "string 5"string 6", $i_3$, $j_3$, $k_3$, "string 7"string 8", $i_4$,$j_4$,$k_4$, "string 9 "string 10", $i_5$,$j_5$,$k_5$ | | String 2 is substituted for string 1 on $i_1$ lines within the character position limits defined by $j_1$ and $k_1$; this is repeated for up to five string pairs, each pair having its own repetition factor and character limit; the command may not exceed one line in length |
| TOP | T | Position at top of file |
| TOP R | | Reassign line numbers and position EDIT at top of file |

# APPENDIX E

# CONFIGURATION EQUATIONS

(Note: See Section 5.2 for the questions Q1, Q2, etc. referred to in the equations)

Equation 1

$$Q2 = Q4 + 1$$

Equation 2

$$Q3 = Q1 + N + 1$$

where

$$N = \frac{B\$TM - 64}{Q13} + \frac{512}{Q13} + \frac{512}{Q13}$$

= number of blocks occupied by BOS core image.

The value of B$TM is the address of B$TM taken from the memory map of BOS which has been converted to a decimal value. Each term in the above equation must be rounded up to the nearest integer.

Equation 3

$$Q4 = Q1 + N1$$

where

$$N1 = N + \frac{\text{size of system library in words}}{Q13 - 3}$$

Each of the two terms in the above equation must be rounded up to the nearest integer.

Equation 4

$$Q5 = Q2 + 1 + N2$$

where

N2 is the number of blocks allocated to the $DO scratch area and is calculated as:

$$N2 = \left(\frac{\text{Total number of DO records required}}{DRB}\right) \text{ rounded up to the nearest integer}$$

and DRB is the greatest integral power of 2 which is less than or equal to the integral part of $\frac{Q13}{11}$

i.e. $DRB = 2^P \leqslant \text{Int. P.C.}\left\{\frac{Q13}{11}\right\}$

## APPENDIX F

## REFERENCE DOCUMENTS

The document numbers of the manuals referred to in Section 1.2 are as follows:

| | |
|---|---|
| Programmer's Reference Manual | Doc. No. 42400343401 |
| FORTRAN Translator Manual | Doc. No. 41286103126 |
| DAP-16 and DAP-16 Mod. 2 Manual | Doc. No. 41286384000 |
| EDIT 700 Manual | |
| Debug Listing | Doc. No. 41285284001 |

# APPENDIX G

# KEY-IN LOADER CODINGS

## 1 BOS Magnetic Tape Key - in Loader

The coding for this Key-in Loader is as follows:

| Location | Octal | Instruction | Comments |
|---|---|---|---|
| 'XX001 | 000011 | DXA | |
| 'XX002 | 140040 | CRA | |
| 'XX003 | 010000 | STA 0 | Set location 0 to point to itself |
| 'XX004 | 030211 | OCP '211 | Read Magnetic Tape three characters/word, Unit 1 |
| 'XX005 | 011020 | STA 'XX020 | Save Entry Point (last word) |
| 'XX006 | 131011 | INA '1011 | Input Word from Magnetic Tape Unit 1 |
| 'XX007 | 003013 | JMP 'XX013 | Not Ready |
| 'XX010 | 110000 | STA* 0 | Store Word (first word is low address) |
| 'XX011 | 024000 | IRS 0 | Tally Pointer |
| 'XX012 | 003005 | JMP 'XX005 | Continue |
| 'XX013 | 070111 | SKS '111 | Skip if Reading Complete (not busy) |
| 'XX014 | 003006 | JMP 'XX006 | Try again (busy) |
| 'XX015 | 070211 | SKS '211 | Skip if no Error |
| 'XX016 | 000000 | HLT | Error Halt |
| 'XX017 | 103020 | JMP*'XX020 | Enter Bootstrap |

The above code should be entered into locations 'XX001 - 'XX017 where XX is the sector into which the code is inserted. It is convenient to use XX = 34.

## 2 BOS Disc Key-in Loader

The coding for this is as follows:

| Location | Octal | Instruction | Comments |
|---|---|---|---|
| '1 | 072007 | LDX '7 | Default load address → X |
| '2 | 032020 | STX '20 | Set DMC Start Address |
| '3 | 000213 | IAB/EXA | Record Number →A, set EXTENDED addressing mode |
| '4 | 010021 | STA '21 | Set dummy DMC End of Range |
| '5 | 030025 | OCP '25 | Seek Cylinder 0 |
| '6 | 170025 | OTA '25 | Output setup word (will always skip) |
| '7 | 100022 | DAC* '22 | Start Address |
| '10 | 030625 | OCP '625 | Read a Record |
| '11 | 170025 | OTA '25 | Output setup word 1 |
| '12 | 002010 | JMP*-2 | Delay until seek finished |
| '13 | 170025 | OTA '25 | Output setup word 2 |
| '14 | 002013 | JMP*-1 | X |
| '15 | 070425 | SKS '425 | Test for Transfer Complete |
| '16 | 002015 | JMP*-1 | No, keep waiting |
| '17 | 042000 | JMP 0, 1 | Yes, Start Bootstrap Execution |

# APPENDIX H

## FORTRAN RUN-TIME ERRORS

When a FORTRAN run-time error occurs, and continuation following the error is not required, a message of the form 'FE Pq' is output on the ASR. For installations possessing a line printer, a further message is also printed in the form:

FORTRAN RUN-TIME ERROR Pq - JOB ABORTED

The possible error mnemonics are listed in the Table H-1, and in parenthesis following each error mnemonic is the name of the relevant library routine in which the error is generated. When continuation following an error is required, that is, Sense Switch 3 is set, the error mnemonic is indicated on the ASR. It should be noted that continuation is not possible following some error conditions.

**TABLE H-1**
**ERROR MNEMONICS**

| Mnemonic | Meaning | Recovery |
|---|---|---|
| AD (A$66) | Overflow/Underflow in double precision addition/subtraction | Continuation uses the largest positive/ negative double precision value |
| AO (SUB$) | Array element referenced is outside defined array bounds | No recovery |
| CE (E$55) | The absolute value of the complex number to be raised to a complex power is zero | Continuation uses zero |
| DL (DLOG) | A negative or zero argument is being used for double precision logarithmic routines | Continuation uses the argument |
| DR (FIO-VAR) | Value of variable device outside legal device range | Continuation defaults to the ASR |
| DT (DATAN2) | Both arguments in a double precision quotient arctangent calculation are zero | Continuation uses zero |
| DZ (M$22 D$22X) | Zero dividend specified or underflow for real divide | Continuation uses an undefined value unless high speed arithmetic option in use when zero returned |
| EF (I/O drivers) | End of file detected | Continuation ignores end of file |
| EQ (A$81) | During double precision exponentiation, overflow occurred when multiplying the double precision accumulator by a power of two | Continuation uses an undefined value |

TABLE H-1 (Continued)

| Mnemonic | Meaning | Recovery |
|---|---|---|
| ET (MAG-IOL) | End of tape mark encountered | Ignore end of tape condition |
| EX (EXP) | During real exponentiation, exponent overflow occurred | Continuation uses the maximum positive real value |
| FE (F$IO) | Error in format statement | Continuation undefined |
| FM (MAG-IOL) | File mark encountered | Ignore file mark |
| GO (F$GA) | The control variable for an assigned GO TO did not match a list entry | No recovery |
| ID (I/O drivers | Action illegal for device | Ignore statement except for READ/ WRITE using variable devices, when ASR actioned by default |
| II (E$11) | Error in raising an integer to an integer power | Continuation uses either, maximum positive integer value if overflow and power even,or zero raised to a negative power; or maximum negative integer value if overflow and power odd |
| IM (M$11 M$11X) | Overflow or underflow in integer multiplication | Continuation uses either maximum positive or maximum negative integer value |
| IN (F$IO) | The last character input is not compatible with the format statement | List element set to zero |
| IZ (D$11 D$11X) | Integer division by zero | Continuation uses the maximum positive or negative integer value, the sign reflecting the sign of the divisor |
|  | Integer division of -32768/-1 attempted | Continuation uses +32767 |
| LG (ALOG ALOGX) | Logarithm of a negative or zero argument | Continuation uses an undefined value |
| MD (A$66 A$66X) | Overflow/Underflow occurred during double precision multiply/divide | Continuation uses the maximum positive or negative double precision value |
| PA (F$ER) | PAUSE statement encountered | Continuation actions next statement |
| PE (MAGIOL) | Parity error | Ignore record with parity error and read next record |
| PZ (A$66 A$66X) | Attempt to divide a double precision argument by zero | Continuation uses the maximum positive or negative double precision value, the sign reflecting the sign of the divisor |

TABLE H-1 (Continued)

| Mnemonic | Meaning | Recovery |
|---|---|---|
| RA (Random files) | Random file range error (see section 7.4.4) | No Recovery |
| RC (Random files) | Incompatible use of random file (see section 7.4.4) | No Recovery |
| RI (C$21) | Conversion from real to integer caused overflow | Continuation uses an undefined value |
| RN (Random files) | File referenced either undefined or non-random (see section 7.4.4) | No Recovery |
| SA (A$22 A$22X) | Overflow occurred during real add or subtract | Continuation uses an undefined value |
| SD (M$22) | The real divisor in a real division operation was not normalised. Overflow on real multiplication | Continuation uses an undefined value |
| SM (M$22 D$22X M$22X) | Arithmetic overflow during real multiply or divide | Continuation uses on appropriate maximum value |
| SQ (SQRT SQRTX) | The argument for a real square-root was negative | Continuation uses an undefined value |
| ST (F$ER) | STOP statement encountered | No recovery |
| ZZ (F$ER) | Path at END of FORTRAN program - default STOP | No recovery |