Burroughs

B 80
MCP MEMORY DUMP

INFORMATION MANUAL

Printed in U.S. America

September 1979

Burroughs believes that the software described in this manual is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, can be accepted for any consequence arising out of the use of this material, including loss of profit, indirect, special, or consequential damages. There are no warranties which extend beyond the program specification.

The Customer should exercise care to assure that use of the software will be in full compliance with laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change. Revisions may be issued from time to time to advise of changes and/or additions.

Correspondence regarding this document should be addressed directly to :

The Manager, Systems Software Support,
Technical Information Organization,
Burroughs Machines Ltd.,
Cumbernauld, G68 0BN,
Glasgow, Scotland.

LIST OF EFFECTIVE PAGES

Page	Issue
Title	Original
ii thru iii	Original
iv	Blank
v thru xiii	Original
xiv	Blank
1–1 thru 1–2	Original
2–1 thru 2–3	Original
2–4	Blank
3–1 thru 3–13	Original
3–14	Blank
4–1 thru 4–4	Original
5–1 thru 5–15	Original
5–16	Blank
6–1 thru 6–19	Original
6–20	Blank
7–1 thru 7–24	Original
8–1 thru 8–30	Original
9–1 thru 9–15	Original
9–16	Blank
10–1 thru 10–26	Original
A–1 thru A–2	Original

CONTENTS

CONTENTS (cont.)

CONTENTS (cont.)

CONTENTS (cont.)

CONTENTS (cont.)

SECTION 1

1. INTRODUCTION

This manual provides information on the B80 CMS MCP, with specific reference to the 3.01 release. It is intended for systems software support personnel, for use when analysing problems and as a guide to the analysis of memory dumps.

The taking of memory dumps and the use of PMB80 to print out selected information is explained. The output of PMB80 is discussed in detail.

Section 4 recommends a method of approach to a systems problem on the CMS B80. The subsequent sections explain the memory organisation, and the main features of the MCP which help towards a general understanding of the system. This is necessary in order to understand a memory dump.

Section 10 presents details on disk organisation which apply to all CMS systems.

The analysis of datacomm areas and the investigation of problems in the B80 Stand Alone Utility are not discussed.

The codes used in the tables to specify the format of MCP maps are explained in Table 1.1.1. The bit numbering convention followed, is that the most significant bit of a byte is numbered 7 and the least significant bit is numbered 0.

The Appendix gives an index of terms used in the body of this manual manual, and explains the meaning of the commonly-used acronyms.

This information should be used by all persons involved in the diagnosis of CMS B80 system problems and the support of the system software.

TABLE 1.1.1 - FORMAT CODES USED IN TABLES

ABBREV.	FORMAT CODE	MEANING
A	ASCII	information stored in ASCII collating sequence
E	EBCDIC	information stored in EBCDIC collating sequence
BCD	BCD	information stored in Binary Coded Decimal format
B	CINARY	information stored in binary
BR	BYTE REVERSED	binary information stored byte reversed most significant byte at highest memory address (on the right)
AA	ABSOLUTE ADDRESS	field contains an absolute memory address
RA	RELATIVE ADDRESS	a memory address relative to some base
SR	SELF RELATIVE	a memory address relative to the field
I	INDEX	a byte index into a field or map
S	SUBSCRIPT	a subscript in a table
L	LABEL	the field name does not reference a field of data but is used to label a memory address
NOTE	NOTE n	refer to the note at the base of the table
TABLE	TABLE n.n.n	refer to the appropriate table in this newsletter

FORMAT CODES USED IN TABLES

CONTENTS - FIGURES

CONTENTS - TABLES

CONTENTS - TABLES (CONT.)

CONTENTS - TABLES (CONT.)

SECTION 2

2. TAKING MEMORY DUMPS

It is very important that a memory dump is taken each time an unexplained failure of the MCP occurs. It is equally important that the dump is taken properly and contains useful information. This section clarifies the memory dump procedures.

Memory dumps may be taken on either cassette or disk. After the system has initialised (either automatically, or by using the initialise button inside the cabinet), the system is in the initial state with PK1 and PK2 lit. If PK2 is pressed, the ROM bootstrap loads the disk bootstrap to memory. PKs 3, 4, 5 & 6 are then enabled. If PK4 is pressed, then the contents of memory may be dumped to cassette; PK5 will cause the contents of memory to be dumped to disk.

The contents of a memory dump are only useful if the MCP had previously been running. Dumps taken after the Stand Alone Utilities have been running are of no use. Also, if PK1 is pressed when PK1 & PK2 are enabled in the initial state, then memory is cleared and memory dump taken after this is of no use.

If an error condition arises during the memory dump routine then a pattern of PK-lights will be lit indicating the error (see tables 2.1.1 and 2.2.1). The error should be corrected, the system initialised again, and the memory dump re-attempted. The memory dump routine destroys the contents of only a very few memory locations (see Table 6.1.1) and repeated attempts at dumping the contents of memory may be made without destroying further information.

2.1 MEMORY DUMP TO CASSETTE

If PK4 is pressed while the system is in the bootstrap-loaded state, then the bootstrap routine attempts to dump the contents of memory to cassette. The numeric keyboard is enabled and a drive number (1-4) must be keyed. The number 1 indicates the drive referenced by the mnemonic CTA by the MCP, and so on.

2.1 MEMORY DUMP TO CASSETTE (CONT.)

If the memory dump routine encouters an error condition, then an indication of the fault is given on the key-board lights (see Table 2.1.1). The error should be corrected, the system initialised again, and a second attempt should be made to dump the contents of memory.

A memory dump cassette has the format of a single file CMS tape with a label of MEMDUMP/MEMORY and a record and block size of 256 bytes.

2.2 MEMORY DUMP TO DISK

If PK5 is pressed while the system is in the bootstrap-loaded state, then the bootstrap routine attempts to dump the contents of memory to disk. The memory dump routine does not create a file on disk, but uses a file already present. A search is made on disk for a file called MEMDUMP and then the contents of memory are written to this file until either the end of memory is reached or the disk file is full.

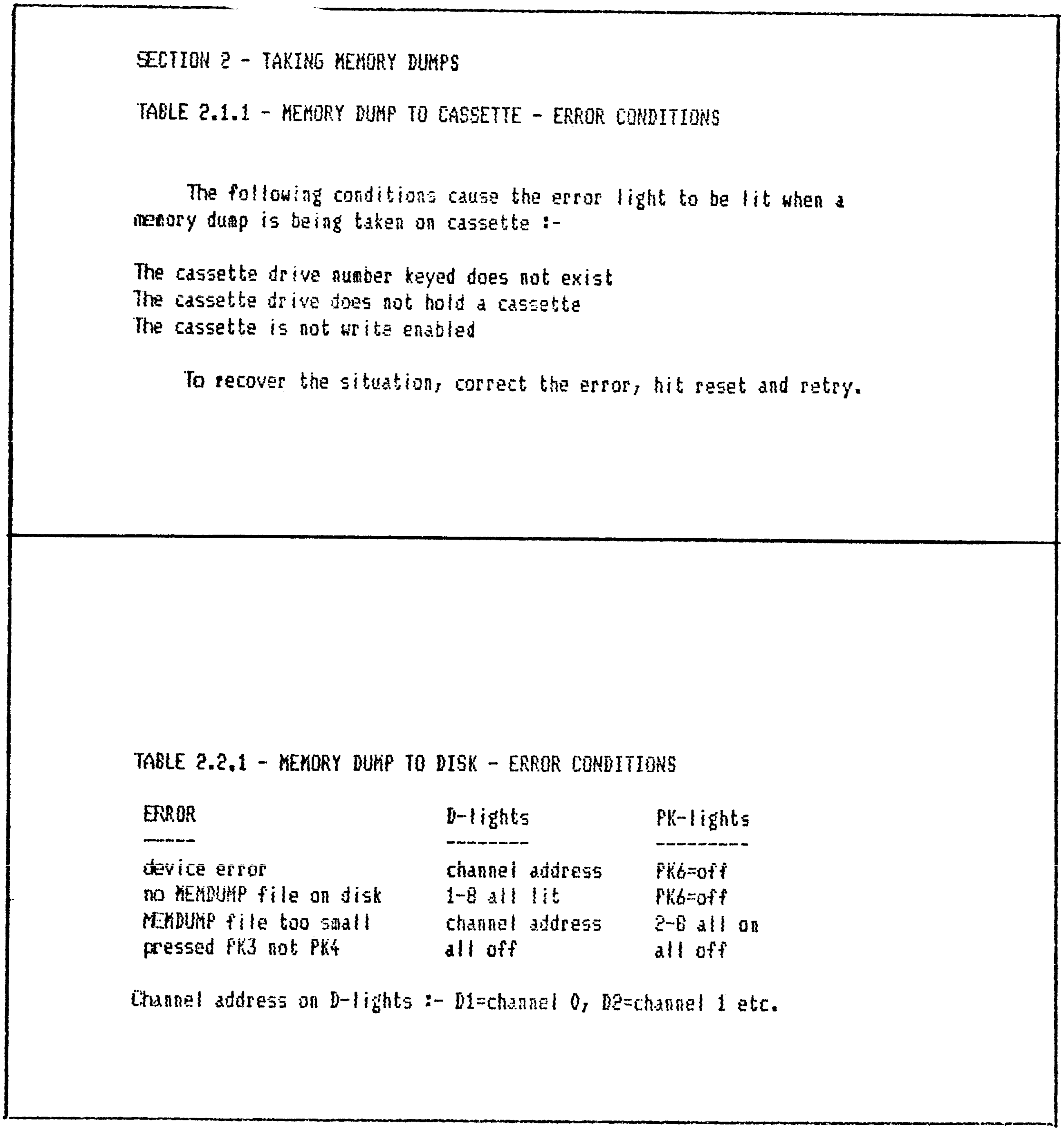
The MEMDUMP file is created on disk by the utility GEN.DUMPFL. It has a record and block size of 180 bytes, and a default filesize large enough to hold 65KB of memory. A larger MEMDUMP file may be created by using an initiating message with GEN.DUMPFL. For example: "GEN.DUMPFL 128" will create a MEMDUMP file large enough to hold 128 KB of memory.

If the memory dump routine encounters an error condition, then an indication of the fault is given on the key-board lights (see Table 2.2.1). The error should be corrected, the system initialised again, and a second attempt should be made to dump the contents of memory.

If there is more than one disk on the system which holds a MEMDUMP file, then confusion may arise about which file contains the latest dump. To avoid this confusion, it is recommended that a disk is set aside which contains a MEMDUMP file of the required size.

Before a memory dump is taken, all other disks should be removed and this disk should be loaded.

Before this disk is re-used, the dump should be analysed or the MEMDUMP file copied to another disk for subsequent analysis.



SECTION 2 - TAKING MEMORY DUMPS

TABLE 2.1.1 - MEMORY DUMP TO CASSETTE - ERROR CONDITIONS

The following conditions cause the error light to be lit when a memory dump is being taken on cassette :-

The cassette drive number keyed does not exist
The cassette drive does not hold a cassette
The cassette is not write enabled

To recover the situation, correct the error, hit reset and retry.

TABLE 2.2.1 - MEMORY DUMP TO DISK - ERROR CONDITIONS

ERROR	D-lights	PK-lights
device error	channel address	PK6=off
no MEMDUMP file on disk	1-8 all lit	PK6=off
MEMDUMP file too small	channel address	2-8 all on
pressed PK3 not PK4	all off	all off

Channel address on D-lights :- D1=channel 0, D2=channel 1 etc.

MEMORY DUMP TO CASSETTE - ERROR CONDITIONS .and
MEMORY DUMP TO DISK - ERROR CONDITIONS

SECTION 3

3. PRINTING MEMORY DUMPS

It is possible to list a memory dump file in hexadecimal and perform the analysis by hand. This however, is a laborious approach, and is not usually required. The CMS B80 system includes a memory dump analyser called PMB80.

PMB80 is a sophisticated tool for selecting and printing information from a system dump file. It also performs some dump analysing functions by highlighting suspected errors, but it is best to confirm these errors by checking the data.

This section presents the features of PMB80 used in memory dump analysis.

3.1 PMB80 AND ITS WORKFILES

The system dump analyse utility consists of the program PMB80 and four reference files PMBHELP, PMBERROR, PMBM.nnnnn and PMBO.nnnnn.

Each major release of the B80 system contains all of these files. In addition, if the format or location of MCP tables (known as MCP "maps") changes with a new version of the MCP, then the Map and Offset files PMBM.nnnnn and PMBO.nnnnn are re-released with the MCP with a new value of nnnnn to identify the MCP version.

The PMBHELP file contains the syntax of the PMB80 commands, and is used to provide the HELP function.

The PMBERROR file contains the messages output by the dump analyser. It may be listed by the LIST utility to provide a list of all possible messages.

The PMBM.nnnnn file holds the format and names of the field of the MCP Maps. The PMBO.nnnnn file holds the memory Offsets of these maps.

3. PRINTING MEMORY DUMPS (CONT.)

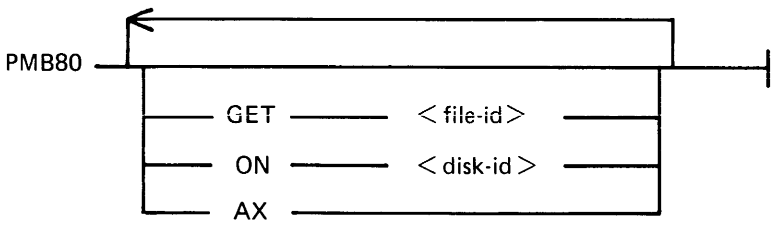
3.1 PMB80 AND ITS WORKFILES (CONT.)

The nnnnn represents the mark, level and patch numbers of the corresponding MCP code file. For example with the 3.01 MCP, the value of nnnnn is 30100. PMB80 can analyse memory dumps of different levels of the MCP. It selects the appropriate Map and Offset files with reference to the field VERSION of map INTERGLBL (Table 7.1.1).

If an attempt to execute PMB80 results in "NO FILE" messages for the Map and Offset files with unrecognisable names, the the VERSION field is likely to be corrupt. This could be a first indication that the memory dump does not contain useful information.

3.2 STARTING PMB80 - THE INITIATING MESSAGE

The utility can be executed with a number of options, selected by the initiating message as follows:



The options provide the following features:

GET - This option is used to specify the file-id of a disk memory dump other than "MEMDUMP" (the default value).

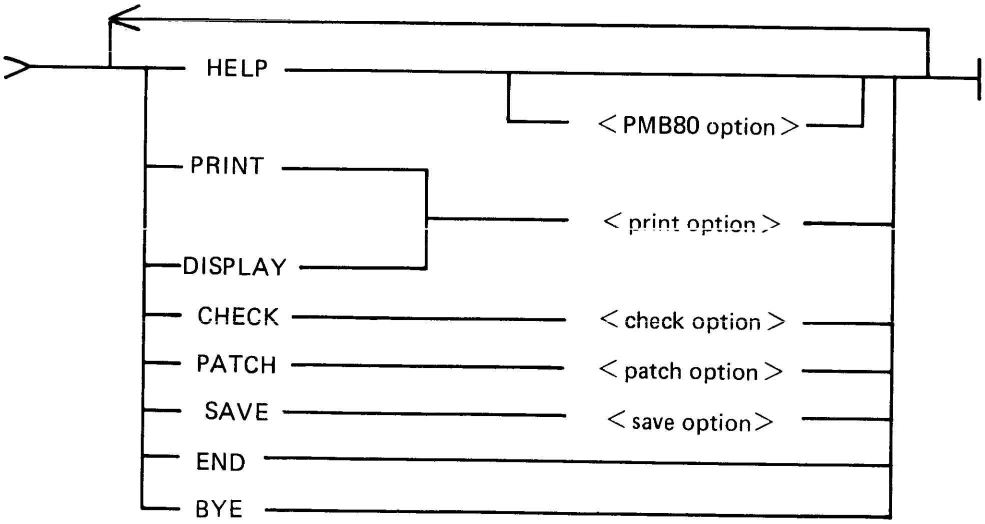
ON - This option is used to specify the disk-id of the disk containing the memory dump file, if other than the system disk.

AX - This option instructs PMB80 to communicate with the operator via DISPLAYS and ACCEPTS. If this option is not specified, then a CONSOLE file is used.

If a disk file called MEMDUMP and a cassette file labelled MEMDUMP/MEMORY are both present, then PMB80 takes the disk file as input.

3.3 PMB80 OPTIONS AVAILABLE

The utility does not print or analyse automatically. The functions of PMB80 are provided on command from the operator. The options available are as follows:



3.3.1 THE HELP OPTION

The HELP option of PMB80 lists the PMB80 options available. If a particular PMB80 option is included (e.g. HELP PRINT) then the syntax and explanation of the various functions of that option are given. Functions can be described in further detail by specifying other options, (e.g. HELP SLICE will give details of the PRINT SLICE function).

3.3.2 THE PRINT AND DISPLAY OPTIONS

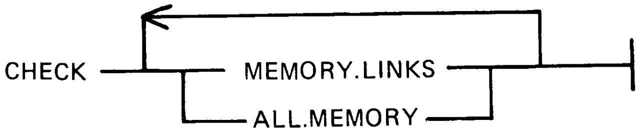
These options control the printing and analysis of selected information.

The PRINT option is the most useful option of PMB80 and is explained in greater detail in section 3.4.

The DISPLAY option provides the same features as the PRINT option, but uses the SPO instead of a PRINTER.

3.3.3 THE CHECK OPTION

This option has the following syntax:



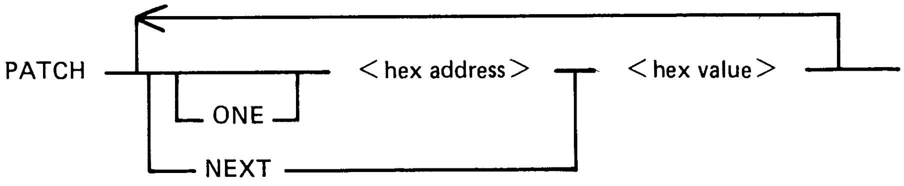
This command instructs PMB80 to analyse the contents of the memory dump and report on any errors encountered.

The MEMORY.LINKS option causes PMB80 to check the structure of overlayable memory (see section 6.4.1). The ALL.MEMORY option instructs PMB80 to check the complete memory structure, and some fields are also checked for valid contents.

It should be noted, however, that it is not feasible for PMB80 to perform a complete analysis of all possible faults. Therefore this option may report an error where no real error exists. Consequently, all faults reported by PMB80 should be verified by checking the data concerned.

3.3.4 THE PATCH OPTION

This option has the following syntax:



3. PRINTING MEMORY DUMPS (CONT.)

3.3 PMB80 OPTIONS AVAILABLE (CONT.)

3.3.4 THE PATCH OPTION (CONT.)

ONE - specifies that the patch is to be performed on page one (extended memory).

NEXT - specifies that this patch is to follow directly after the previous patch.

<hex-address> - is four hexadecimal digits with no delimiters.

<hex-value> - is from one to 16 hexadecimal digits.

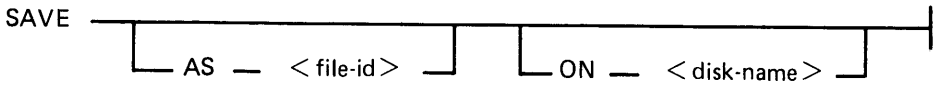
Sometimes a memory dump contains corrupt information which limits the extent of useful analysis that can be performed by PMB80.

The PATCH option enables the operator to correct invalid areas of memory, enabling PMB80 to continue its analysis. The patches are applied to an internal PMB80 work file, leaving the original dump file unchanged.

The modified dump file may be saved using the SAVE option (see section 3.3.5). If this action has been taken, details should be documented when reporting a problem, and both the original and patched dump files should be submitted.

3.3.5 THE SAVE OPTION

The syntax for this option is:



AS - This option allows the operator to specify the file-id of the memory dump file which is saved. The default value is "MEMDUMP".

ON - This option allows the operator to specify the disk-id of the disk on which the dump file is to be saved. If this option is not used, then the dump file will be saved on the system disk.

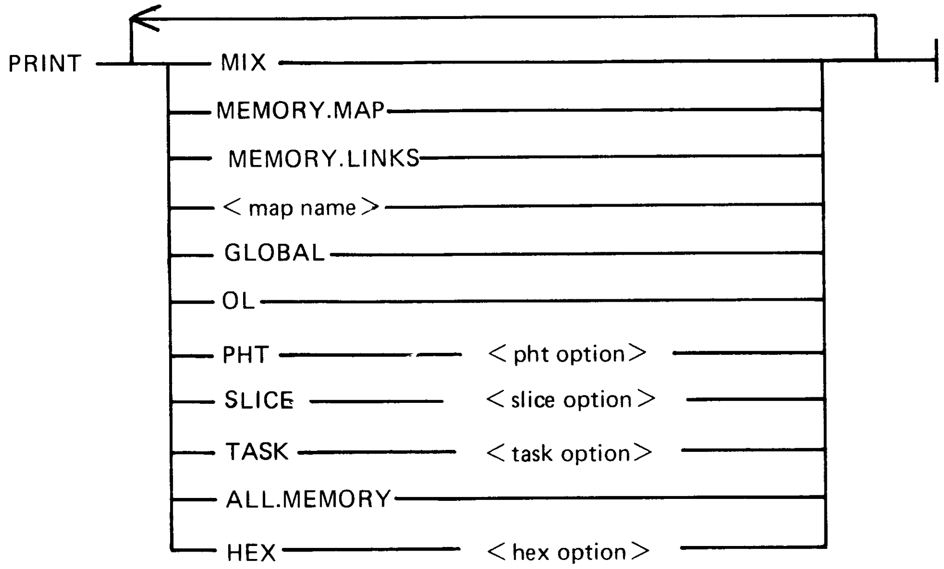
3.3.6 THE END AND BYE OPTIONS

Either of these options will cause PMB80 to go to end of job (EOJ).

3. PRINTING MEMORY DUMPS (CONT.)

3.4 THE PRINT OPTIONS

The options available within the print command are as follows:



When starting analysis of a memory dump, it is recommended that items are printed in the order in which they are given above.

The MIX option provides information on the tasks that were running. If a hardware problems is suspected, then the OL and PHT options provide information on the state of peripheral configuration. The following sections explain the PRINT options in more detail.

3.4.1 THE PRINT MIX OPTION

This option provides a selective analysis of information contained in the global MCP map GLBLM (Table 7.6.1) and the TASK.TABLE (Tables 8.2.1 and 8.2.2).

A list is given of the tasks running at the time of the dump, what status these tasks were in, and which tasks (if any) held locks to non re-entrant MCP code. Since the TASK.TABLE is an overlayable segment of the Bailiff Slice (Slice 0) the task names may not be available on the memory dump.

3.4.2 THE PRINT MEMORY.MAP OPTION

This option provides an analysis of the layout of memory. The contents of the five sections (RESIDENT, LOCKED, OVERLAYABLE, PHT, EXTENDED) are listed in the order in which they occur. If any faults such as overlapping fields are detected, then a message is printed.

It is always advisable to check such error messages carefully. Sometimes the overlap is not a real error (for example: Zero length segments may legally overlap; sometimes the LOADER (slice 15) locates its segments within its CONTROL STACK; the INITIALISE (slice 18) slice descriptor may overlap since it is not used; etc.).

The analysis of the locked and overlayable areas is performed through the SAT, slice descriptors and Data Segment Tables (see Section 6.3). Any error reported in this part of the analysis implies an error in one of these structures.

3.4.3 THE PRINT MEMORY.LINKS OPTION

This option analyses the layout of the OVERLAYABLE AREA of memory. The analysis is performed through the memory link mechanism described in Section 6.4. Any error reported with this option indicates the corruption of this structure.

If the system was performing virtual memory operation when the dump was taken, then the memory links are likely to be in a transient state and no conclusion can be drawn from apparent errors. In this condition, the PRINT MIX option will show the VMLOCK to be held by a task.

3.4.4 THE PRINT GLOBAL OR PRINT<map name> OPTION

This option causes MCP tables to be printed in a map format. A map is a "template" which the MCP uses to define its data structures. The <map name>s available are:

GWA : Global Work Area. See Table 7.1.1 - INTERGLBL MAP, and Section 7.1.

3. PRINTINT MEMORY DUMPS (CONT.)

3.4 THE PRINT OPTIONS (CONT.)

3.4.4 THE PRINT GLOBAL OR PRINT map name OPTION (CONT.)

PHDMP : Peripheral Handling dump area. See Table 7.2.1 and Section 7.2.

DIAGNOSTICS : See Table 7.4.1 - DIAGCBUF MAP and Section 7.4.

VMWA : Virtual Memory Work Area. See Table 7.5.1 and Section 7.5.

ESCT : Execution Scan Table. See Table 7.6.1 - GLBLM MAP, and Section 7.6.

SCL : Slice Address Table. See Table 6.2.1 SATM Map and Section 6.2.

TASK.TABLE : See Tables 8.1.1 and 8.1.2 and Section 8.1. This table is overlayable and therefore may not be present in memory.

C.TABLE : Configuration Table. See Table 9.1.1 - CT MAP, and Section 9.1. This table is overlayable and therefore may not be present in memory.

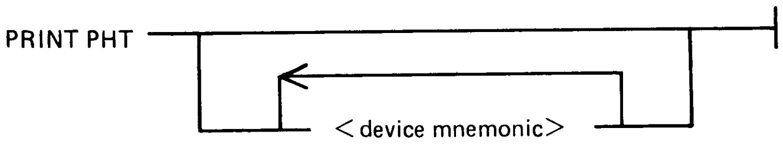
The PRINT GLOBAL option will cause the global MCP tables (INTERGLBL, PHDMP, VERSIONINFO, DIAGCBUF, VMWA, GLBLM, SCL, CT.INFO, SAT) to be printed.

3.4.5 THE PRINT OL OPTION

This option provides a selective analysis of information contained in the C.TABLE (Table 9.1.1). This includes the peripherals configured on the system,any media loaded on the peripherals, and the status of the peripherals. The C.TABLE is overlayable and so this analysis may not be possible.

3.4.6 THE PRINT PHT OPTION

This option enables the printing of the Peripheral Handling Table for selected device types present on the system. The syntax is:



3. PRINTING MEMORY DUMPS (CONT.)

3.4 THE PRINT OPTIONS (CONT.)

3.4.6 THE PRINT PHT OPTION (CONT.)

Valid <device mnemonic>s include:

DF	-	Fixed Disk
DK	-	Disk Cartridge
DM	-	Burroughs Super Mini Disk
LP	-	Line Printer
SP	-	Serial Printer
SS	-	Self Scan
KB	-	Keyboard
CT	-	Cassette
CX	-	Channel Expander
DI	-	Industry Compatible Mini disk
ADC	-	Async. Data Comm
SDC	-	Sync. Data Comm

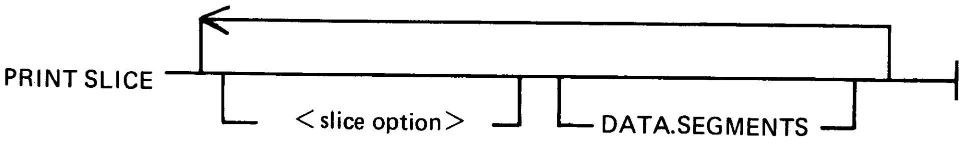
For each <device mnemonic> specified, all the PHTs for devices of that type are printed. If no <device mnemonic> is entered, then all the PHTs in the dump are printed.

In addition to printing the PHTs, this option prints the I-O queues, queued descriptors, and keyboard, console and SPO buffers (where applicable). Sections 9.2 and 9.3 discuss these tables in more detail.

3.4.7 THE PRINT SLICE OPTION

This command prints the contents of selected slices (see section 6.3 for a description of memory slices).

The syntax is:



DATA.SEGMENTS - This option prints the contents of the data segments in the overlayable area. If this option is not used, then only the contents of the locked slice will be printed.

3.4 THE PRINT OPTIONS (CONT.)

3.4.7 THE PRINT SLICE OPTION (CONT.)

Values for <slice option> include:

O-48 : Any number in this range refers to a slice
See Table 6.2.1 - SATM MAP

SUSN : (Slice 12) Super Utility TCB (data)

OPENCLOSE : (Slice 17) File Open & Close MCP code

INITIALISE : (Slice 18) Warmstart MCP code

SPO : (Slice 19) SPO printing MCP code

LPDDR : (Slice 20) Line printer DDR code

CASSDDR : (Slice 21) Cassette DDR code

DISKDDR : (Slice 22) Disk DDR code

SENDDR : (Slice 23) 60 cps Serial Printer DDR code

KBDDR : (Slice 24) Keyboard DDR code

SCREENSN : (Slice 25) Self-Scan/CRT DDR code

ADCDDR : (Slice 26) Async. Data Comm DDR code

SDCDDR : (Slice 27) Sync. Data Comm DDR code

DCCH : (Slice 28) Data Comm Communicate Handler MCP code.

CONSOLE : (Slice 29) Console Communicate Handler MCP code

PANDDR : (Slice 20) 180 cps and 120 cps Serial Printer DDR code

INXS : (Slice 31) Indexed file Comm Handler MCP code

ICMDDDR : (Slice 33) ICMD DDR code

CONBUFSN : (Slice 35) Console Buffer data slice

SCLBUFSN : (Slice 36) SCL Buffer data slice

3.4.7 THE PRINT SLICE OPTION (CONT.)

If no <slice option> is specified, then all the slices will be printed in the order in which they reside in the locked area.

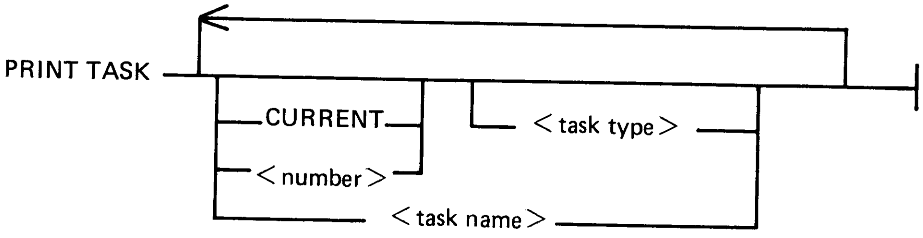
Items which may be included in the analysis of a loaded slice are:

MAP RS	- Slice Descriptor (see Table 6.3.1)
FIELD S.I.W.A.	- S—Interpreter Work Area (in program TCBs only) (see Tables 8.2.1 and 8.3.1).
FIELD CCBPA2	- COP table (COBOL/RPG CCBs only)
FIELD CONTROL STACK	- TCBs only (see table 7.4.2).
MAP SD	- Segment Descriptor (see Table 6.4.1)

If the DATA.SEGMENTS option is selected then further items discussed in Section 8 may be encountered.

3.4.8 THE PRINT TASK OPTION

This command prints the contents of a Task Control Block (TCB) (see Section 8). The syntax is:



3. PRINTING MEMORY DUMPS (CONT.)

3.4 THE PRINT OPTIONS (CONT.)

3.4.8 THE PRINT TASK OPTION (CONT.)

Options for <task name> include:

BAILIFF	:	Slice 0, MCP code
MCS	:	Slice 13, MPLII s-code
NDL	:	Slice 14, NDL s-code
SCL	:	Slice 15, MCP code (including LOADER)
LOADER	:	Slice 15, MCP code (including SCL)

<number> : must be the mix number of a task in the range 0-15

CURRENT : causes the printing of the task which was executing prior to the dump being taken. This is the task referenced by field EICT of map GLBLM (Table 7.6.1), and may be part of the cause of the dump.

<task type> enables further analysis of the S-Interpreter Work Area (SIWA). If the option is not selected then the SIWA is printed as a single block of data.

Options available include:

COBOL RPG	: :	COBOL S-machine (see Table 8.2.1)
MPLII BIL	: :	BIL S-machine (see Table 8.3.1)
SORT	:	Sortintrins microcode (refer to MCP listing)
NDL	:	NDL S-machine

If no option is specified for the PRINT TASK command, then all tasks in the dump are printed.

In addition to printing the TCB, the PRINT TASK option prints the associated Program Control Block (PCB) and Interpreter Control Block (ICB), wherever applicable. See section 8 for further details of these task structures.

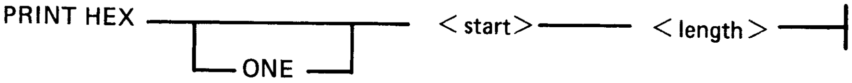
3.4.9 THE PRINT.ALL MEMORY OPTION

This option prints the entire contents of the memory dump. It is printed in a format equivalent to a command of PRINT GLOBAL PHT SLICE. Memory not in use is not printed. Although this option prints all of memory it is not recommended; selective options may considerably reduce the volume of printed output.

NOTE: It is not sufficient to include only this printout when submitting problem reports. Useful information may be contained in memory areas not in use. This is especially true if memory links are corrupt. When submitting a problem report containing a memory dump, the complete dump on cassette, disk or load/dump tape should therefore be included. An accompanying listing of the PRINT MIX GLOBAL OL PHT option is useful for initial analysis.

3.4.10 THE PRINT HEX OPTION

This option provides the operator with a printout of selected areas of the dump file. The option has the syntax:



ONE : This option selects page one (extended) memory for printing.

<start> : This parameter must be four hexadecimal digits long (no delimiters) and indicates the memory address of the first byte to be printed.

<length> : This parameter must also be four hexadecimal digits long. It indicates the number of bytes to be printed.

The contents of the memory dump are printed in hexadecimal and ASCII format.

SECTION 4

4. INITIAL DUMP ANALYSIS

This section discusses the approach to analysing memory dumps. Firstly, the type of problem must be identified. Once this is done, some general analysis should be performed which may lead to deeper analysis of selected parts of the dump.

4.1 IDENTIFYING THE PROBLEM

Problems can be partially analysed by considering the symptoms. The operating environment (what tasks/programs were running), and the configuration environment (what devices were in use) may indicate the general nature of the problem. First of all, the following categories should be differentiated.

4.1.1 ALL PK-LIGHTS FLASHING

This is known as the memory parity condition. It occurs when the processor fetches a byte from memory which has a parity error. In this situation a complete memory dump cannot be taken since the problem will occur again when the dump is being taken. However, a memory dump taken to cassette will proceed up to the bad memory address and therefore locate the problem.

Note that memory dump files on disk are not initialised to any specific pattern; consequently, the end of the dumped information cannot easily be located on a MEMDUMP file.

The memory parity condition may also be caused by a system software failure. The ROM at the low address end of memory contains a deliberate permanent parity error as a check on interpreter failures when accessing beyond segment boundaries. Therefore, if no memory hardware fault can be located, the current task should be analysed (see Section 4.2.2).

4.1.2 SOME PK-LIGHTS FLASHING

When the MCP detects an irrecoverable error condition, it sets PK-lights 17 to 24 flashing in a pattern which identifies the problem. Refer to section 5.4 for details.

If analysis does not reveal faulty hardware or corrupt media, further analysis may be performed to isolate the environmental conditions under which the problem occurred. If more than one dump with similar symptoms exist, analysis often gives a clue to the cause.

4.1.3 INITIALISATION TO PK-LIGHTS 1 AND 2

This is caused by the execution of the microcode @000000@, which passes control to the start of memory. The first thing to look for in a memory dump taken after this condition, is corrupted memory. If there appears to be no corruption, then a general analysis as discussed in section 4.2 should be followed.

4.1.4 NO RESPONSE TO KEYBOARD INPUT (NO ACTIVITY)

In this condition, the MCP gives no response to depressing any key, including the Ready Request-Key. This may be due to a peripheral problem, or the MCP might be in a tight loop. The global diagnostic area (see Section 7.4) should distinguish the two cases.

Since the keyboard is associated with the console printer, a printer jam will cause this symptom. The PHT of the serial printer should be examined if this is a suspected cause of the dump.

4.1.5 NO RESPONSE TO KEYBOARD INPUT (D-LIGHTS FLICKERING)

It may not be possible to enter data or obtain a response from depressing the Ready Request-Key, but the system may still look "busy" because of D-light activity. This problem is often caused by system thrashing. This can result from an attempt to execute too many programs in insufficient memory, or an attempt to execute a program with exceptionally large segments. In this case, the organisation and contents of memory should be examined.

4.1.6 NO PROGRAM RESPONSE TO "GO" OR "DS" INPUT

In this condition, no action appears to be taken by a program when a "GO" or "DS" message is entered for that program, although all other activity appears to be normal. This is usually caused by the program hanging on a dedicated peripheral (i.e. not disk). In this case the task status should be examined in the memory dump.

4.2 PRELIMINARY INVESTIGATION

It is recommended that the first PMB80 analysis of a dump should be PRINT MIX MEMORY.MAP MEMORY.LINKS GLOBAL OL PHT (see section 3). This provides a wide range of information without producing an excessive amount of printed output.

4.2.1 HARDWARE STATUS

The OL and PHT options of PMB80 print this information. It is recommended that a check is always made on the status of the devices involved in any problem (see section 9.2). An invalid status may not necessarily cause a system problem, but unexpected conditions may arise which cause a failure.

The device status in the CT (PRINT OL option) is a logical status as seen by the MCP; this may not exactly correspond with the physical hardware status. The PHT holds the last status received from the channel controller.

4.2.2 MEMORY ORGANISATION AND CONTENTS

The PRINT MEMORY.MAP MEMORY.LINKS option of PMB80 provides a complete analysis of the layout of memory. Section 6 describes the format of memory with no error conditions. All reports of overlapping areas in memory should be investigated. If the Virtual Memory lock is held by a task (section 7.6.2), then overlayable memory may contain an invalid link.

Shortage of overlayable memory may be the cause of a number of performance problems, especially system thrashing. If a dump is taken because of poor performance, then the amount of overlayable memory should be checked. For a rough guide, at least 10 KB is required as "free space" to enable virtual memory to perform efficiently. Large program code or data segments can cause similar problems. Although it is not possible to specify an exact limit, as a rough guide it may be stated that user program segments of 4K bytes are too large, especially on the B80 in a multi-programming environment.

Section 7 discusses the global maps; these should be checked for valid contents.

The PRINT PHT option of PMB80 analyses the I-O descriptor queues. Since the queue heads lie at the high address end of memory, an incomplete memory dump will provide an incorrect analysis of this structure.

4.2.3 MCP STATUS

The status of the MCP when the memory dump was taken can be obtained from an analysis of the global maps (section 7). The diagnostic area in particular is designed with this purpose in mind. The analysis of the GLBLM MAP and TASK.TABLE (PRINT MIX option of PMB80) provides a general overview of the task environment.

4.2.4 USER TASK STATUS

Task structure analysis is provided by the PRINT TASK option of PMB80, and discussed in section 8. To avoid producing a large amount of printed output without much relevant information, it is recommended that the current task is analysed first. The current task is the task running when the memory dump was taken. The most important information, other than the integrity of the task structure, is the communicate parameter area. This indicates the latest call on MCP routines (open file, read etc.) that was requested from the task.

SECTION 5

5. SYSTEM REGISTERS AND TRACE DIAGNOSTICS

The B80 MCP is written in microcode, consequently extensive reference is made throughout this newsletter to the hardware registers of the B80 processor. This section documents the format and usual contents of these registers.

The B80 MCP provides a general trace facility controlled by the GT command. The trace is used for system fault diagnosis. Even when the trace is not in operation, a memory dump includes a historical buffer of one byte containing trace diagnostic information (see section 7.4). Sections 5.2 through 5.4 describe trace diagnostics.

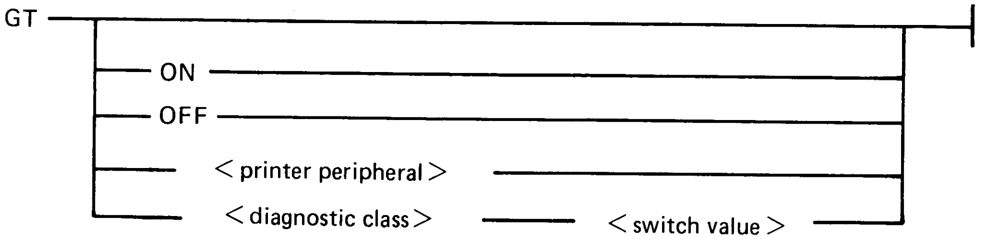
5.1 SYSTEM REGISTERS

Table 5.1.1 shows the format of information in the processor registers, and their possible contents. Although a memory dump does not necessarily contain the contents of the registers at the time of the dump, the following points should be noted.

The PHDMP MAP (Table 7.2.1) stores the current contents of the registers after two types of event. When a physical interrupt is received from an I-O channel, the processor executes the Master Interrupt Processor (MIP) section of the MCP. The registers are dumped on entry to the MIP in order that processing can resume, after the interrupt has been serviced, from the point of interruption. The registers are also dumped to the PHDMP MAP if the general trace is running (see section 5.2) or the system failed with a trace diagnostic (section 5.4).

5.2 THE GT COMMAND

The general trace command is an MCP intrinsic which displays various diagnostic information either on the system console printer or on a line printer. The format is:



5.2 THE GT COMMAND (CONT.)

The ON and OFF options switch the trace on or off respectively. The <printer peripheral> selects the device to print any trace messages. If this device is currently in use, the trace messages are interleaved with other printed output. Table 5.2.1 shows the <diagnostic class>es which are further explained in the sections which follow.

The MCP contains calls to the trace routine throughout the code. However, trace diagnostics are printed only when two conditions are fulfilled; (a) when the trace is switched on, and (b) when the level of the switch value for that class is less than or equal to the level of the trace point. The <switch value> consists of two hexadecimal digits, with the following meanings:

first digit : switch value for register diagnostics
second digit : switch value for memory diagnostics

In addition to providing a printed trace of selected trace points, the general trace displays a one-byte diagnostic (the contents of the AD register) on PK lights 17 to 24 (PK17 corresponding to the most significant bit and PK24 the least significant bit. If the PK light is lit, the corresponding bit value is 1. These lights display the last trace point encountered if the trace option is switched on and the keyboard is not enabled for input.

5.3 TRACE DIAGNOSTICS

Trace information provided by the B80 MCP consists of three parts:

register diagnostics
memory diagnostics
one-byte diagnostics

Register diagnostics are made to a selected printer and the contents of the registers are entered in PHDMP (table 7.2.1) when the trace is switched on and the switch value is lower than the trace point value for that class. Memory diagnostics are made to a selected printer when the same conditions are satisfied. One-byte diagnostics are displayed on PK lights 17 to 24 whenever the trace is switched on and the keyboard is not input enabled. The most recent one-byte diagnostic processed by the MCP can be found in the field DIAGCIRC of map DIAGCBUF (Table 7.4.1) regardless of the state of the trace or switch values.

5. SYSTEM REGISTERS AND TRACE DIAGNOSTICS (CONT.)

5.3 TRACE DIAGNOSTICS (CONT.)

Printed register diagnostics have the following format:

```
register  AD  BO  B1FL B32   J    K    L
contents  XX  XX  XXXX XXXX  XXXX XXXX XXXX

          M1   M2   WR   X        Y        AD,ESCT
          XXXX XXXX XXXX XXXXXXXX XXXXXXXX XXXX
```

(the actual print is on one line). AD contains the one-byte diagnostic consisting of two hexadecimal digits. The first digit represents the diagnostic class (see Table 5.2.1) the second digit represents the diagnostic severity value which is compared with the switch value set by the GT command. ESCT is the one byte task-id (Table 7.6.2) of the currently executing task (as displayed on the keyboard D-lights).

A general rule on the interpretation of the diagnostic severity value is the following:

O (zero) is the least important
B is used when a module is entered (eg: DB = VM module entry)
E is used when a module is exited (eg: EE= EPAR module exit)
F is associated with an error condition (not necessarily fatal)

Actual details vary according to the coding of the various MCP modules.

5.3.1 FILE OPEN AND CLOSE - CLASS O

These diagnostics are issued by the OPENCLOSE MCP slice 17, which performs operations requested by a task via the communicate mechanism. There are so many possible interpretations of each severity value that it is not possible to give a guide to interpretation of register diagnostics beyond OB = entry and OE = exit. Memory diagnostics in this class often print a DFH, an FIB or an FPB.

If class O diagnostics appear in DIAGCIRC it is possible that the problem is related to opening or closing a file. Possible faults include media corruption (disk directory, program S-code files), or memory corruption of the FPB. These possibilities may be checked from a memory dump or by running utilities such as KA and CHECK.DISK. The task which initiated the operation may hold one of the openclose locks (see section 7.6) and the task's communicate area (in SIWA) will indicate a verb of 01 = open or 02 = close (see table 8.4.1).

5.3.2 INDEXED FILE HANDLING - CLASS 1

These diagnostics are issued by the MCP index communicates slice 31. A value of 1B indicates slice entry, and 1E indicates slice exit. This slice performs operations requested by a task via the communicate mechanism.

5.3.3 ACCEPT/DISPLAY/DATE/TIME - CLASS 2

These diagnostics are issued by the MCP SPO slice 19 which processes communicates with a verb in the range @10@ - @20@ (see Table 8.4.1).

5.3.4 INTRINSIC UTILITIES - CLASSES 3-7

The SORTINTRINS microcode uses these classes.

5.3.5 AUTOMATIC VOLUME RECOGNITION - CLASSES 8-9

Class 8 diagnostic is used by the general AVR routine (OPENCLOSE segment 18); class 9 is used specifically by the routine which "tidies up" a disk directory at AVR time (OPENCLOSE segment 19). If AVR entries are found in DIAGCBUF of a memory dump, it is quite likely that the problem has been caused by faulty hardware or media. In addition to recognising newly loaded media, the AVR routine is used when certain exceptional events occur on hardware channels.

5.3.6 DISK SPACE ALLOCATION/DE-ALLOCATION - CLASS A

This class of diagnostics is issued by OPENCLOSE segments 20 and 21 which perform the disk space allocation and de-allocation functions respectively. (see Table 5.3.6).

5.3.7 INTERPRETERS - CLASS B

This class of diagnostics is reserved for the interpreter microcode. However, only the BILINTERP (MPLII program interpreter) uses the facility extensively.

This class of diagnostics is very useful tracing an MPL program. See table 5.3.7 for the details. Note that BB and BE do not indicate interpreter entry and exit in this case.

5.3.8 COMMUNICATE HANDLING - CLASS C

The global MCP routine for decoding all communicates from tasks uses this class of diagnostics. See Table 5.3.8.

5.3.9 VIRTUAL MEMORY - CLASS D

The Virtual Memory (VM) routine uses this class of diagnostics. A trace of the virtual memory operations is not generally needed as a lot of detailed analysis is involved. However, when a disk problem occurs the VM routine is often the first part of the MCP to detect the error. This is the reason why the DF failure diagnostic (section 5.4.2) is often encountered.

5.3.10 TASK CONTROL (EPAR) - CLASS E

There are not many trace points with this class. Most important is the EE diagnostic, issued when the task schedules are exited. This indicates which task has been entered and which code is currently executing (see section 7.4).

5.3.11 INPUT/OUTPUT QUEUE HANDLING - CLASS F

This class of trace diagnostic shows the queueing of I-O descriptors, and the waiting of tasks on these I-O operations. In the register diagnostics the registers hold the fields of the I-O descriptor being queued. (See Table 5.3.11).

5.4 FAILURE DIAGNOSTICS

When the MCP discovers that an irrecoverable error has occurred, it issues a failure diagnostic. On previous systems this was a printed trace diagnostic. On release 3.01 a failure diagnostic is a pattern of flashing PK lights. PK lights 17 to 24 show the one-byte diagnostic of the failure with PKs 17 to 20 showing the diagnostic class.

When this situation occurs, a memory dump should be taken (see section 2). The PHDMP MAP (Table 7.2.1) contains the contents of the processor registers at the time of the failure. The value of the one-byte diagnostic indicates the rough nature of the problem; the contents of the registers give more specific information.

5.4.1 AC, AD (PK patterns 10101100, 10101101)

AC and AD on PK lights 17-24 indicate a fatal error detected while allocating or de-allocating disk file space respectively. The M1 register holds the memory address of the last disk I-O descriptor and BO holds IODFL (see Table 9.3.4). This problem is usually caused by disk hardware or disk media problems.

5.4.2 DF (PK pattern 11011111)

More than one type of error is detected by the VM subsystem. These are distinguished by the contents of the M1 register (see Table 5.3.9).

If M1=1111 or 5555, an I-O error has been encountered on a virtual memory operation. Refer to the VM I-O descriptor (field VMIOD of MAP VMWA, Table 7.5.1). This indicates a problem with the disk drive or disk media.

If M2=2222, then the MCP has attempted an invalid "put segment" operation. If M1=FFFF then an unconditional "get" operation has failed (not caused by an I-O error). If either of these problems persists after the system (including user programs) has been recovered from backup copies, then a fault may exist in the MCP and should be reported.

5.4.3 EF (PK pattern 11101111)

Different errors identified by this failure diagnostic are distinguished by the contents of the M1 register (see Table 5.3.10). If any of these problems persist after the system has been recovered from backup copies, then the problems should be reported.

5.4.4 FF (PK pattern 11111111)

This error diagnostic is used to show errors in peripheral handling. The WR register identifies the device type involved (see Table 5.3.11). The PHT entries for the device should be checked for a bad status condition (section 9.2). This may identify a hardware fault. If no hardware fault is located, the problem should be reported.

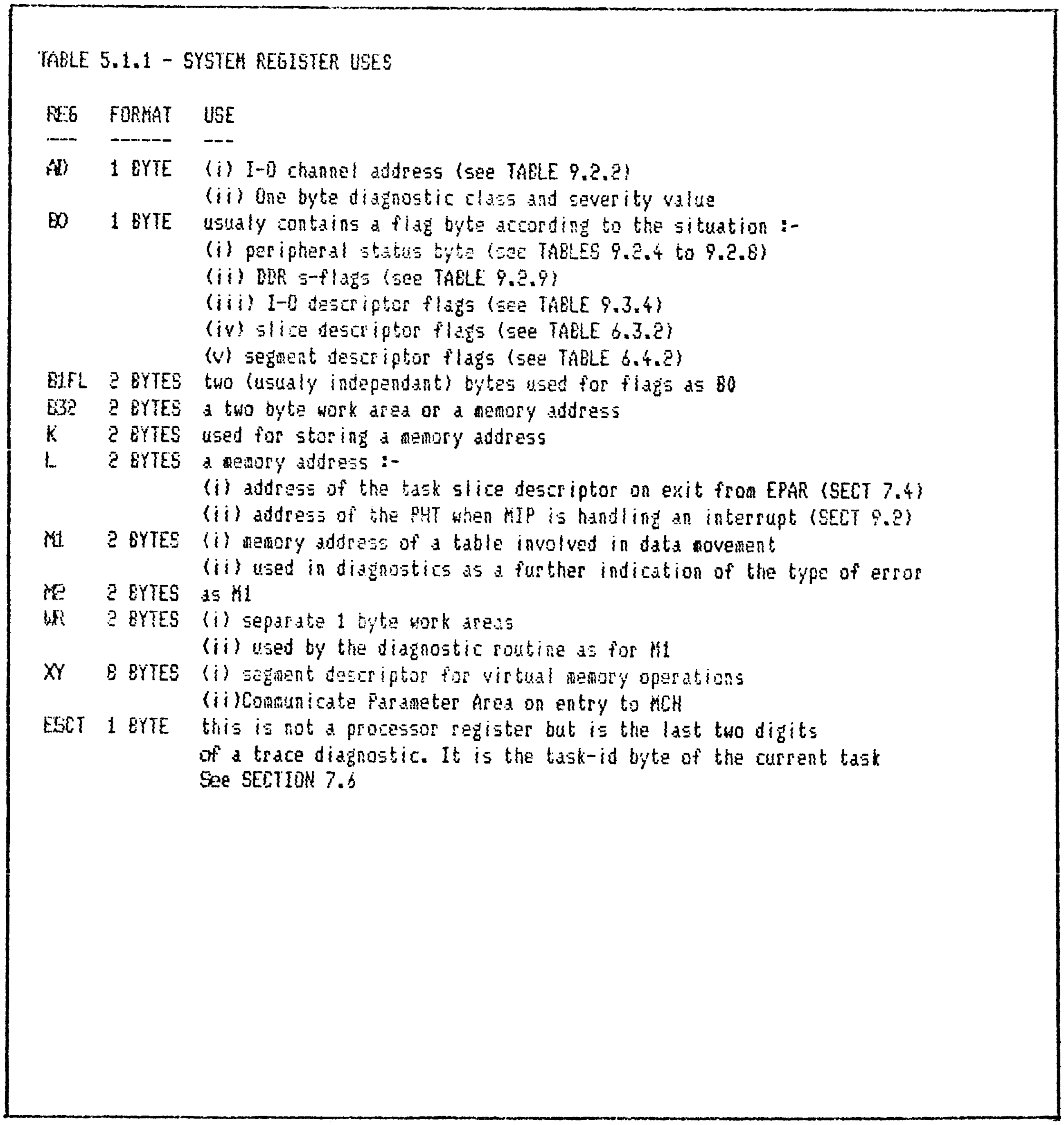


TABLE 5.1.1 - SYSTEM REGISTER USES

REG	FORMAT	USE
A0	1 BYTE	(i) I-O channel address (see TABLE 9.2.2) (ii) One byte diagnostic class and severity value
B0	1 BYTE	usualy contains a flag byte according to the situation :- (i) peripheral status byte (see TABLES 9.2.4 to 9.2.8) (ii) DDR s-flags (see TABLE 9.2.9) (iii) I-O descriptor flags (see TABLE 9.3.4) (iv) slice descriptor flags (see TABLE 6.3.2) (v) segment descriptor flags (see TABLE 6.4.2)
B1FL	2 BYTES	two (usualy independant) bytes used for flags as B0
B32	2 BYTES	a two byte work area or a memory address
K	2 BYTES	used for storing a memory address
L	2 BYTES	a memory address :- (i) address of the task slice descriptor on exit from EPAR (SECT 7.4) (ii) address of the PHT when MIP is handling an interrupt (SECT 9.2)
M1	2 BYTES	(i) memory address of a table involved in data movement (ii) used in diagnostics as a further indication of the type of error
M2	2 BYTES	as M1
WR	2 BYTES	(i) separate 1 byte work areas (ii) used by the diagnostic routine as for M1
XY	8 BYTES	(i) segment descriptor for virtual memory operations (ii)Communicate Parameter Area on entry to MCH
ESCT	1 BYTE	this is not a processor register but is the last two digits of a trace diagnostic. It is the task-id byte of the current task See SECTION 7.6

SYSTEM REGISTER USES

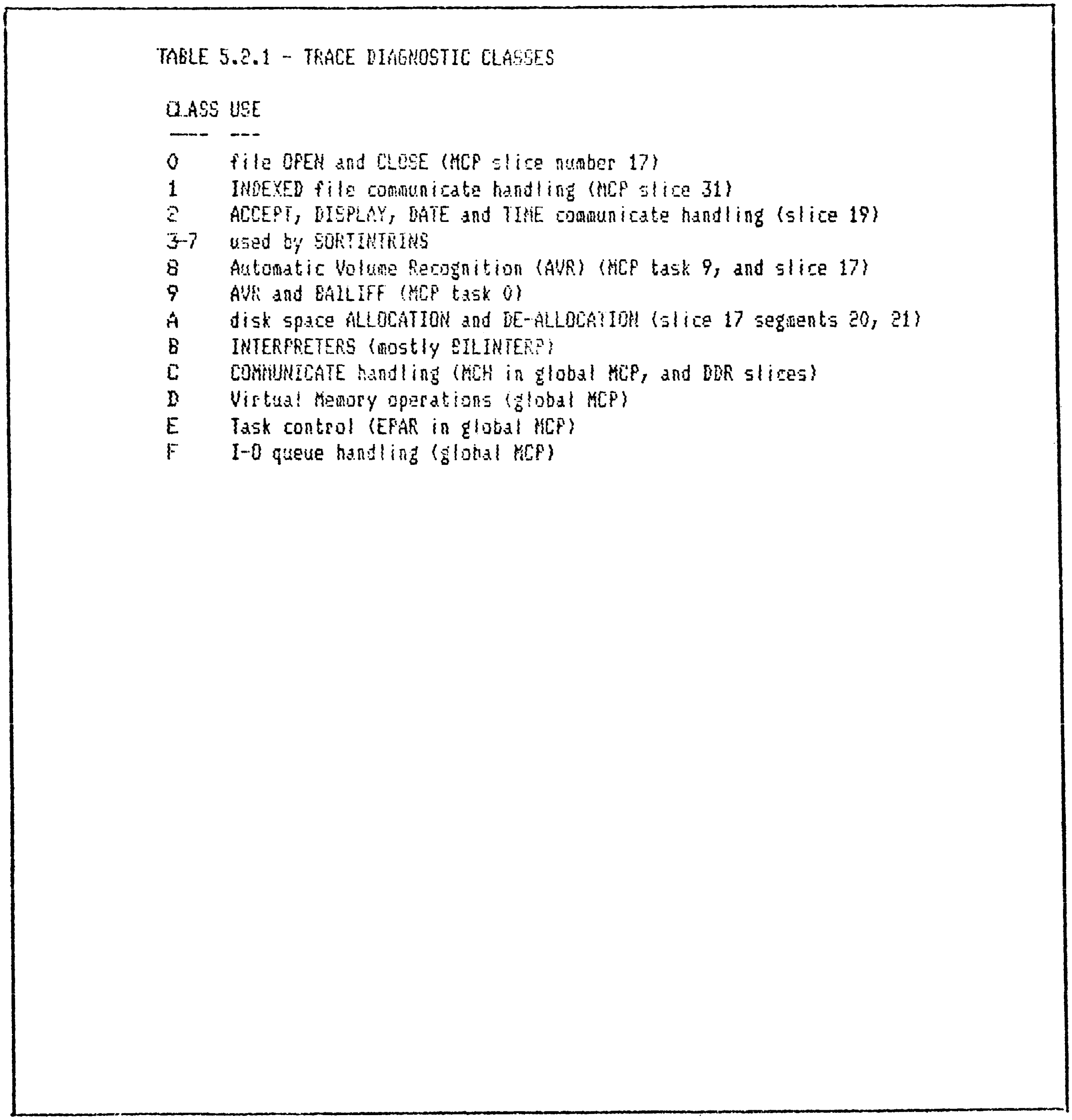


TABLE 5.2.1 - TRACE DIAGNOSTIC CLASSES

CLASS	USE
0	file OPEN and CLOSE (MCP slice number 17)
1	INDEXED file communicate handling (MCP slice 31)
2	ACCEPT, DISPLAY, DATE and TIME communicate handling (slice 19)
3-7	used by SORTINTRINS
8	Automatic Volume Recognition (AVR) (MCP task 9, and slice 17)
9	AVR and BAILIFF (MCP task 0)
A	disk space ALLOCATION and DE-ALLOCATION (slice 17 segments 20, 21)
B	INTERPRETERS (mostly SILINTERP)
C	COMMUNICATE handling (MCH in global MCP, and DDR slices)
D	Virtual Memory operations (global MCP)
E	Task control (EPAR in global MCP)
F	I-O queue handling (global MCP)

TRACE DIAGNOSTIC CLASSES

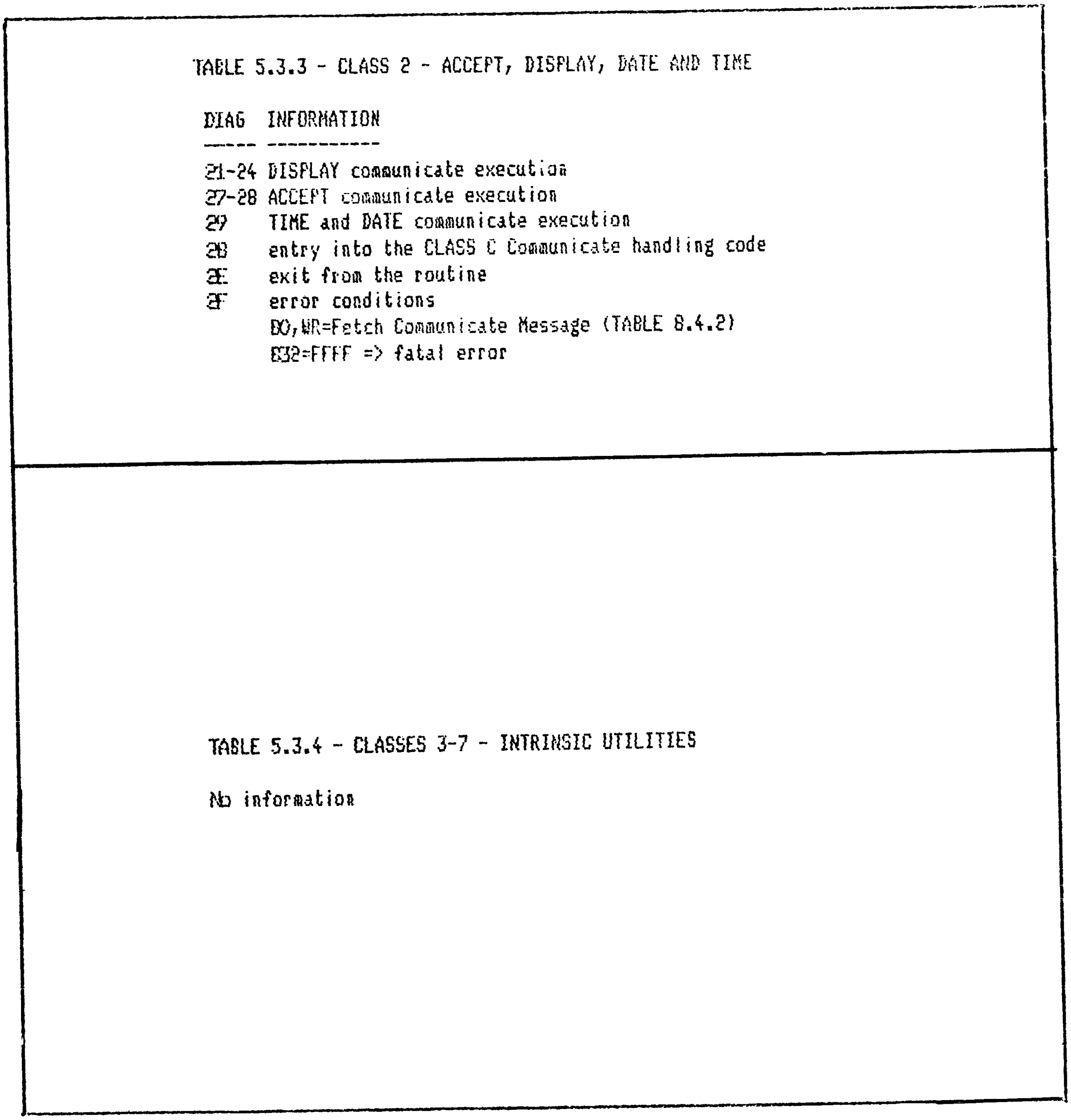
TABLE 5.3.1 - CLASS 0 - FILE OPEN AND CLOSE

DIAG	INFORMATION
0B	entry into one of the open or close routines
0E	exit from one of the open or close routines

TABLE 5.3.2 - CLASS 1 - INDEXED FILE COMMUNICATE HANDLING

DIAG	INFORMATION
10	deleted entry found, or end of area entry found
11	suspending the operation
12	searching through the keys
13	comparing the keys
14	set up overflow region search buffer
15	set up index region search buffer
16	access work area key
17	fill the entry (WRL = key entry size)
18	store the entry (WRL = key entry size)
19	end of the free slot sliding the buffer up
1A	call MIP to queue the I-O descriptor B0=IODDU (E=write, F=read) (TABLE 9.3.3)
1B	entry into the index comms routine B0=WRL=communicate verb, B1=IFLAGS
1E	exit from the index comms routine B1=FILESTATE (TABLE 8.6.3)

CLASS 0 - FILE OPEN AND CLOSE and
CLASS 1 - INDEXED FILE COMMUNICATE HANDLING



```
TABLE 5.3.3 - CLASS 2 - ACCEPT, DISPLAY, DATE AND TIME

 DIAG  INFORMATION
 ----- -----------
 21-24 DISPLAY communicate execution
 27-28 ACCEPT communicate execution
 29    TIME and DATE communicate execution
 2B    entry into the CLASS C Communicate handling code
 2E    exit from the routine
 2F    error conditions
       B0,WR=Fetch Communicate Message (TABLE 8.4.2)
       B32=FFFF => fatal error
```

```
TABLE 5.3.4 - CLASSES 3-7 - INTRINSIC UTILITIES

No information
```

CLASS 2 - ACCEPT, DISPLAY, DATE AND TIME and CLASSES 3-7 - INTRINSIC UTILITIES

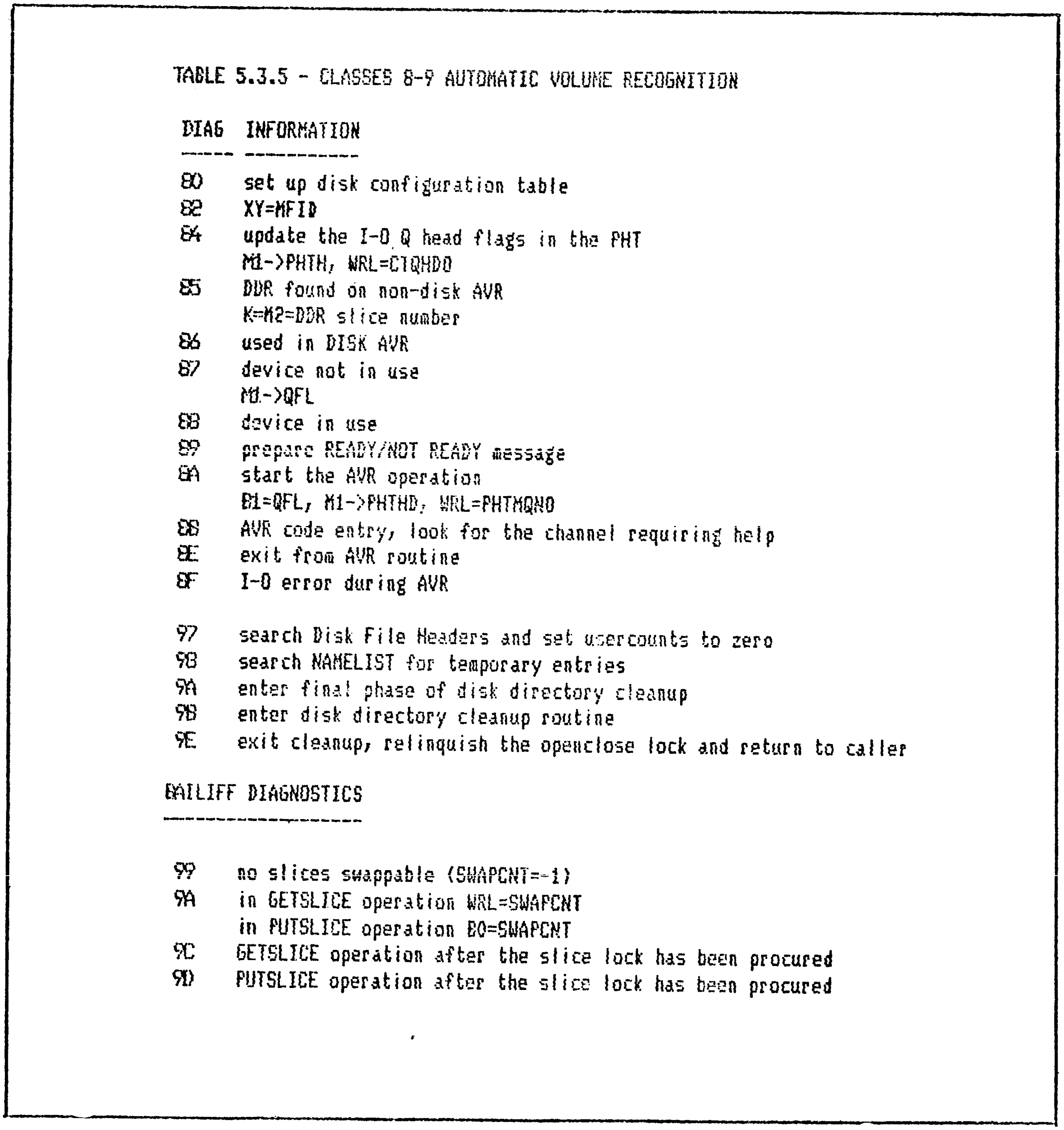


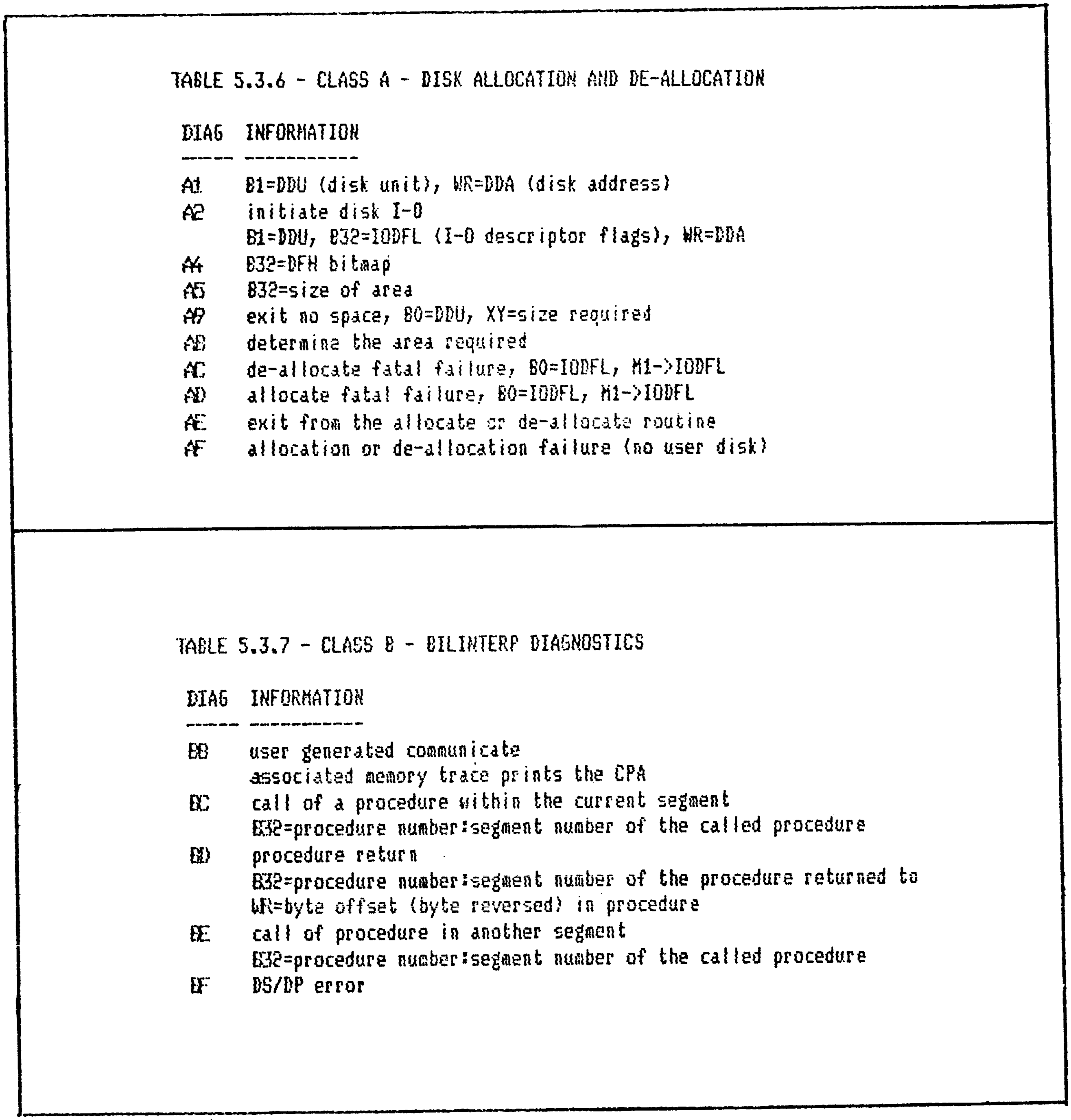
TABLE 5.3.5 - CLASSES 8-9 AUTOMATIC VOLUME RECOGNITION

DIAG	INFORMATION
80	set up disk configuration table
82	XY=MFID
84	update the I-O Q head flags in the PHT M1->PHTH, WRL=C1QHD0
85	DDR found on non-disk AVR K=M2=DDR slice number
86	used in DISK AVR
87	device not in use M1->QFL
88	device in use
89	prepare READY/NOT READY message
8A	start the AVR operation B1=QFL, M1->PHTHD, WRL=PHTMQN0
8B	AVR code entry, look for the channel requiring help
8E	exit from AVR routine
8F	I-O error during AVR
97	search Disk File Headers and set usercounts to zero
98	search NAMELIST for temporary entries
9A	enter final phase of disk directory cleanup
9B	enter disk directory cleanup routine
9E	exit cleanup, relinquish the openclose lock and return to caller

BAILIFF DIAGNOSTICS

99	no slices swappable (SWAPCNT=-1)
9A	in GETSLICE operation WRL=SWAPCNT in PUTSLICE operation B0=SWAPCNT
9C	GETSLICE operation after the slice lock has been procured
9D	PUTSLICE operation after the slice lock has been procured

CLASSES 8-9 - AUTOMATIC VOLUME RECOGNITION



```
TABLE 5.3.6 - CLASS A - DISK ALLOCATION AND DE-ALLOCATION

DIAG  INFORMATION
----- -----------

A1    B1=DDU (disk unit), WR=DDA (disk address)
A2    initiate disk I-O
      B1=DDU, B32=IODFL (I-O descriptor flags), WR=DDA
A4    B32=DFH bitmap
A5    B32=size of area
A9    exit no space, B0=DDU, XY=size required
AB    determine the area required
AC    de-allocate fatal failure, B0=IODFL, M1->IODFL
AD    allocate fatal failure, B0=IODFL, M1->IODFL
AE    exit from the allocate or de-allocate routine
AF    allocation or de-allocation failure (no user disk)
```

```
TABLE 5.3.7 - CLASS B - BILINTERP DIAGNOSTICS

DIAG  INFORMATION
----- -----------

BB    user generated communicate
      associated memory trace prints the CPA
BC    call of a procedure within the current segment
      B32=procedure number:segment number of the called procedure
BD    procedure return
      B32=procedure number:segment number of the procedure returned to
      WR=byte offset (byte reversed) in procedure
BE    call of procedure in another segment
      B32=procedure number:segment number of the called procedure
BF    DS/DP error
```

CLASS A - DISK ALLOCATION AND DE-ALLOCATION and
CLASS B - BILINTERP DIAGNOSTICS

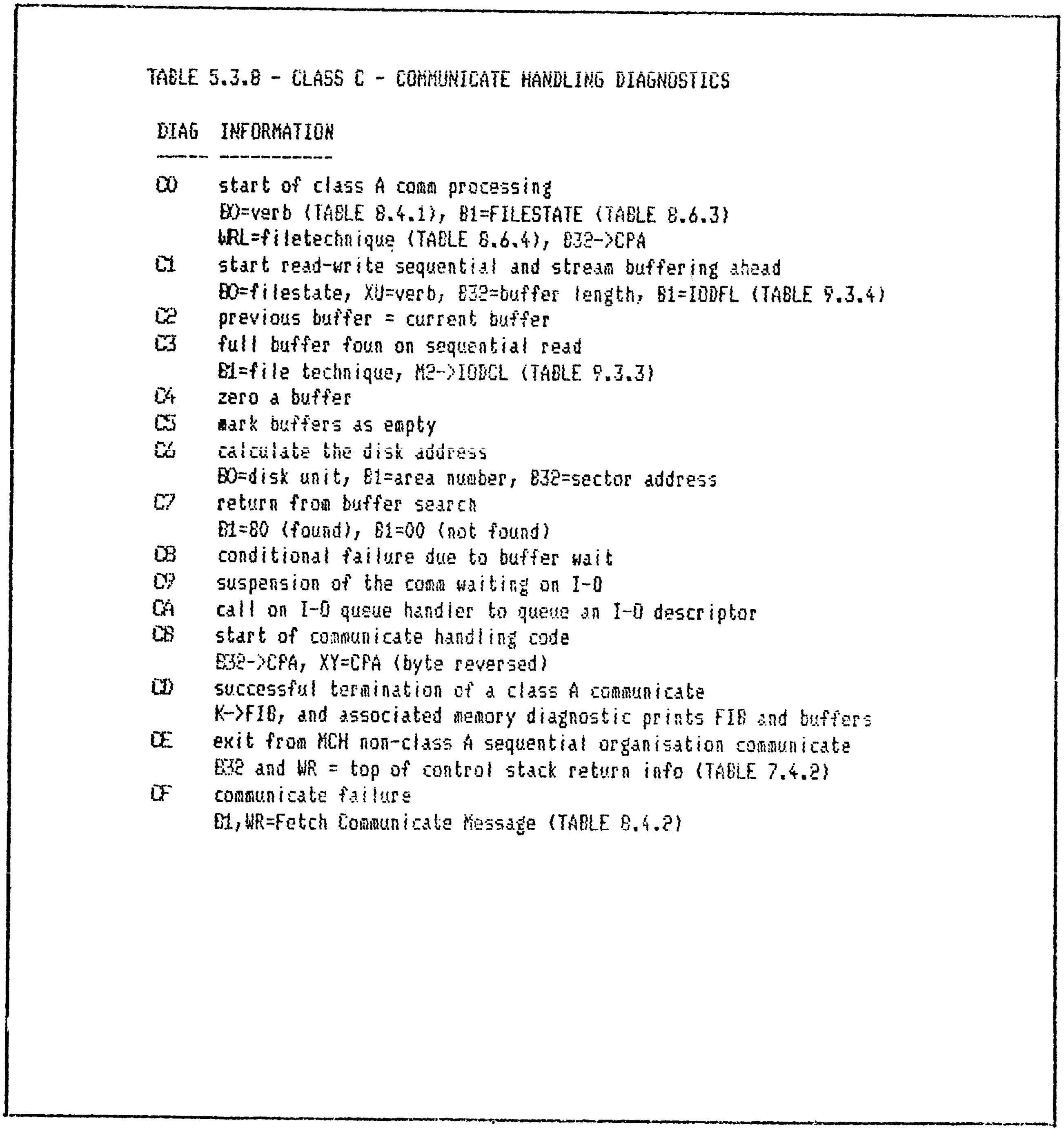


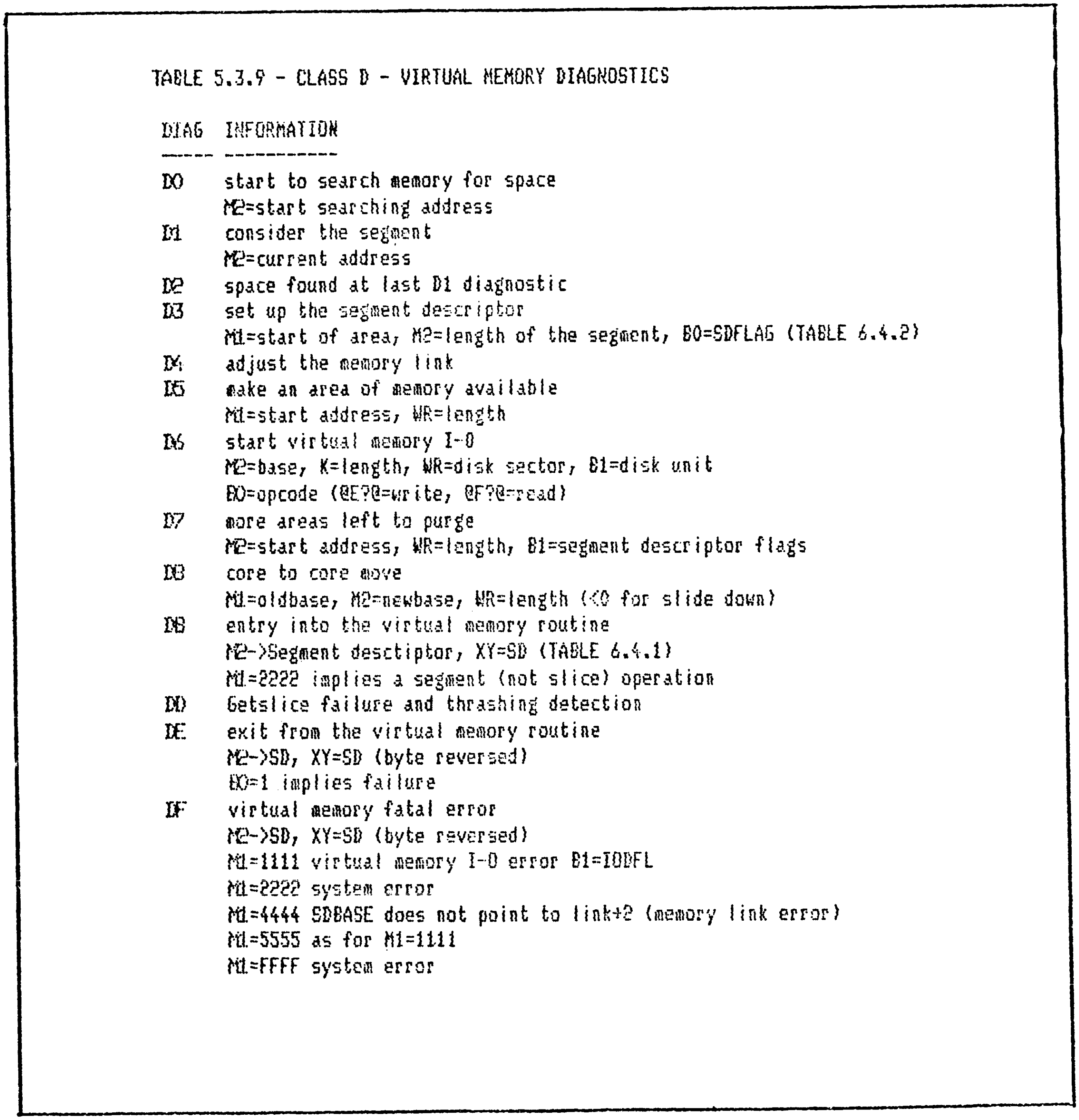
TABLE 5.3.8 - CLASS C - COMMUNICATE HANDLING DIAGNOSTICS

DIAG	INFORMATION

C0	start of class A comm processing
	B0=verb (TABLE 8.4.1), B1=FILESTATE (TABLE 8.6.3)
	WRL=filetechnique (TABLE 8.6.4), B32->CPA

C1	start read-write sequential and stream buffering ahead B0=filestate, XU=verb, B32=buffer length, B1=IODFL (TABLE 9.3.4)
C2	previous buffer = current buffer
C3	full buffer foun on sequential read B1=file technique, M2->IODCL (TABLE 9.3.3)
C4	zero a buffer
C5	mark buffers as empty
C6	calculate the disk address B0=disk unit, B1=area number, B32=sector address
C7	return from buffer search B1=80 (found), B1=00 (not found)
C8	conditional failure due to buffer wait
C9	suspension of the comm waiting on I-O
CA	call on I-O queue handler to queue an I-O descriptor
CB	start of communicate handling code B32->CPA, XY=CPA (byte reversed)
CD	successful termination of a class A communicate K->FIB, and associated memory diagnostic prints FIB and buffers
CE	exit from MCH non-class A sequential organisation communicate B32 and WR = top of control stack return info (TABLE 7.4.2)
CF	communicate failure B1,WR=Fetch Communicate Message (TABLE 8.4.2)

CLASS C - COMMUNICATE HANDLING DIAGNOSTICS



```
TABLE 5.3.9 - CLASS D - VIRTUAL MEMORY DIAGNOSTICS

DIAG  INFORMATION
----- -----------
D0    start to search memory for space
      M2=start searching address
D1    consider the segment
      M2=current address
D2    space found at last D1 diagnostic
D3    set up the segment descriptor
      M1=start of area, M2=length of the segment, B0=SDFLAG (TABLE 6.4.2)
D4    adjust the memory link
D5    make an area of memory available
      M1=start address, WR=length
D6    start virtual memory I-0
      M2=base, K=length, WR=disk sector, B1=disk unit
      B0=opcode (@E?@=write, @F?@=read)
D7    more areas left to purge
      M2=start address, WR=length, B1=segment descriptor flags
D8    core to core move
      M1=oldbase, M2=newbase, WR=length (<0 for slide down)
DB    entry into the virtual memory routine
      M2->Segment desctiptor, XY=SD (TABLE 6.4.1)
      M1=2222 implies a segment (not slice) operation
DD    Getslice failure and thrashing detection
DE    exit from the virtual memory routine
      M2->SD, XY=SD (byte reversed)
      B0=1 implies failure
DF    virtual memory fatal error
      M2->SD, XY=SD (byte reversed)
      M1=1111 virtual memory I-0 error B1=IODFL
      M1=2222 system error
      M1=4444 SDBASE does not point to link+2 (memory link error)
      M1=5555 as for M1=1111
      M1=FFFF system error
```

CLASS D - VIRTUAL MEMORY DIAGNOSTICS

```
TABLE 5.3.10 - CLASS E - TASK CONTROL DIAGNOSTICS

DIAG  INFORMATION
----- -----------
EB    entry into System Control Language decoder
      L->TCB of originator
ED    slice usercount overflow
EE    exit from epar into the current task
      XY=top of stack (TABLE 7.4.2)
      J->microcode segment base, L->slice descriptor, M->top of stack
EF    fatal error
      M1=9999 segment usercount error detected on PUTSEG
      M1=AAAA attempt to use an absent slice
      M1=BBBB error in SCL interpreter
```

```
TABLE 5.3.11 - CLASS F - INPUT-OUTPUT QUEUE HANDLING DIAGNOSTICS

DIAG  INFORMATION
----- -----------
      for all of these diagnostics except FF the registers hold :-
      B0=IODDU, B1=IODFL, B32->IODFL, M1=IODDA, M2=IODBL (see TABLE 9.3.3)
F2    unconditional wait on exit after call from MCH
F3    conditional wait on exit after call from MCH
F4    no wait on exit after non-disk descriptor inserted
F6    unconditional wait after non-disk descriptor inserted
F7    conditional wait after non-disk descriptor inserted
F8    no wait on exit after disk descriptor insert
FA    unconditional wait on exit after disk descriptor insert
FB    conditional wait on exit after disk descriptor insert
FF    fatal DDR error
      M1=0000 invalid disk unit in I-I descriptor B0=IODDU
      WR=BBBB Data comm DDR error
      WR=9999 self scan DDR error
      WR=EEEE disk DDR error
```

CLASS E - TASK CONTROL DIAGNOSTICS and
CLASS F - INPUT-OUTPUT QUEUE HANDLING DIAGNOSTICS

SECTION 6

6. MCP STRUCTURE AND MEMORY ORGANISATION

It is often necessary to know where certain items may be found in memory. Corruption of memory is one of the more common problems encountered in an MCP. On some dumps an area of memory is over-written with invalid information; on other dumps, just one or two critical bytes holding a memory link (address to another item) are wrong. PMB80 may be unable to analyse satisfactorily a MEMDUMP file containing invalid memory links, since it assumes the memory structure is correct. This section provides sufficient information to enable a HEX dump to be analysed. A hex dump is obtained by the PRINT HEX option of PMB80.

6.1 AREAS OF MEMORY AND THEIR USES

The six areas of memory used by B80 3.01 MCP are shown in Figure 6.1.1.

The Read Only Memory (ROM) which has addressed @0000@ to @0FFF@ contains interrupt handling routines. It appears full of binary zeros on a hex dump.

When the bootstrap routine is loaded to memory it destroys the contents of those areas of memory specified in table 6.1.1. These areas do not contain information which is important for dump analysis.

The other five areas are discussed in sections 6.2 to 6.6.

6.2 RESIDENT AREA AND GLOBAL MCP

This contains the global MCP code and data, which does not change location while the system is running (see Figure 6.2.1). This area of memory contains:

Global tables (discussed in section 7)

Global MCP routines:

Master Interrupt Processor (MIP)
Master Communicate Handler (MCH)
Virtual Memory (VM) Handling code

6.2 RESIDENT AREA AND GLOBAL MCP (CONT.)

Global MCP routines:

Task Structure Switching (EPAR)

Slice Address Table (SAT)

Each item is located at a fixed address and the same table or code segment can be found at the same address on all dumps for the same level of MCP. The code routines are not analysed by PMB80. The addresses of the tables may be found from either an MCP listing or a correctly analysed dump.

6.2.1 SLICE ADDRESS TABLE (SAT)

The SAT is a table of memory address of the next section of memory. Each memory address is two bytes long and addresses the first byte of the Slice Descriptor of a slice in the locked area. See Table 6.2.1, SATM MAP, and figure 6.3.1. An analysis of the SAT is obtained by the PRINT SAT option of PMB8O.

6.3 LOCKED AREA AND SLICES

The Locked Area of memory follows directly after the SAT. The first slice (the BAILIFF) is at the same memory address on all dumps of the same version of the MCP.

In the locked area, slices are contiguous with no intervening areas. Slices may be either present or absent (swapped out). If a slice is swapped out, only the slice descriptor remains in memory. When a slice is swapped in or out by the BAILIFF, all other items in the locked area are moved up or down in memory so that all slices remain contiguous.

An analysis of the layout of memory is obtained by the PRINT MEMORY.MAP option of PMB8O.

6.3 LOCKED AREA AND SLICES (Cont'd)

6.3.1 SLICES

A slice is a body of code or data which provides a specific function. Examples of slices are: the MCP routine to handle the SPO (slice 19), the data comm message buffers (slice 34), the data associated with a task (program) in the mix (slices 1-14), or the code associated with a task in the mix (slices 39-49). A slice does not necessarily contain its own data, but rather controls access to its data which is located in the overlayable area of memory. The different types of slices have several features in common.

The first ten bytes of each slice is a slice descriptor (see Table 6.3.1, RS MAP). This descriptor contains information about the status of the slice (SDFLAG, SDUSRS), the size and location of the disk copy (SDLENG, SDDKAD, SDDKSC, and SDUNIT) and the PINK LINK.

The slices are linked together in the order in which they reside in memory. This is not in slice number order, the order of SAT. The PINK LINK is the memory address of the first byte of the next slice descriptor. This scheme of linking the slice descriptors is used to relocate memory slices when slices are swapped in or out.

A slice is either a Code Control Block (CCB) or a Task Control Block (TCB). A CCB may be a Device Dependent Routine (DDR), a Function Dependent Routine (FDR), an Interpreter Control Block (ICB) or a Program Control Block (PCB). A TCB holds the control information for a task.

6.3.2 DEVICE DEPENDENT ROUTINE (DDR)

Each hardware device (disk, cassette, line printer, etc.) supported by the B80 has a section of MCP code which handles all features unique to the device. This code is accessed through a DDR slice. Each different device type has its own DDR with its corresponding slice number, and its entry in the SAT. For example, the DISKDDR is slice number 22 with an entry in SAT at offset 44 (@2C@).

6.3.3 FUNCTION DEPENDENT ROUTINE (FDR)

Part of the MCP code is divided into units each of which performs a specific set of functions. The code for each unit is accessed through a slice. Examples of function dependent routines are the indexed file communicate handler (slice 31), the open/close communicate handler (slice 17), and the warmstart slice (slice 18). In addition to these slices for accessing MCP function-dependent code, there are slices which hold buffers, for example: the data comm message buffers (slice 34).

6.3.4 INTERPRETER CONTROL BLOCK (ICB)

The code for an interpreter is also accessed through the slice mechanism. An interpreter slice, however, does not have its permanently allocated slice number and corresponding entry in the SAT. Interpreter slices are classifed as user slices and take a slice number from the pool of available entries at the top of the SAT. Unfortunately, this makes it difficult to distinguish program code slices from interpreter code slices when analysing dumps. The task structures, (see section 8), allow these slices to be distinguished.

It should be noted that if an interpreter is located in extended memory then it is allocated a permanent slice from the available pool and the slice is located directly following the BAILIFF and AVR slices in the memory map. If the interpreter is not loaded into extended memory at warmstart time, then its slice is created and released whenever programs which require the interpreter are loaded or terminated. In this case the interpreter slice is loaded after a TCB, and resides next to the Program Control Block.

6.3.5 PROGRAM CONTROL BLOCK (PCB)

Program code is accessed through the data segment table located within this locked slice. For COBOL and RPG programs the COP Table is also located within the task's PCB. PCB slices are user slices and take a slice number from the pool of available entries at the top of the SAT.

6.3.6 TASK CONTROL BLOCK (TCB)

The TCB is the slice which holds the control information for a task. Its contents are discussed in detail in section 8.

Each TCB has a slice number which corresponds to the mix number of its task. It addresses the corresponding PCB and ICB, and holds the CONTROL STACK which contains MCP restart information. The data segments for a program are accessed through the Data Segment Table held within the TCB.

6.3.7 DATA SEGMENT TABLE (DST)

A locked slice does not usually hold all of the code or data identified with the slice. Most slices contain a Data Segment Table (DST). The DST is a table of segment descriptors (see table 6.4.1 SD MAP) giving the address of the associated code on disk, and also the memory address if the segment is present in main memory.

The absolute memory address of the DST is held in bytes offset 14 and 15 (reversed) of the locked slice (see Tables 8.2.1 and 8.3.1). All items in the dump that locate other information in memory have now been described.

6.3.8 EXAMPLES OF HOW TO LOCATE ITEMS

Locate data segment 9 of a program running with mix = 4:

a) Find the SAT (Fixed memory address).

b) The address of the TCB slice descriptor is at byte offset 2 x 4 = 8 and is byte reversed.

c) The address of the DST is in bytes 14 and 15 of this slice (reversed), (Tables 8.2.1 and 8.3.1).

d) The segment descriptor is located at byte offset 8 x 9 = 72 in the DST, and is 8 bytes long.

e) The first byte of SD determines whether the segment is in memory, (Table 6.4.2)

f) Bytes offset 1 and 2 (reversed) give the memory address of the segment, (Table 6.4.1).

6.3.8 EXAMPLES OF HOW TO LOCATE ITEMS (CONT.)

Locate the COP Table of a program running with mix = 3 (COBOL/RPG programs only):

a) Find the SAT (Fixed memory address)

b) The address of the TCB slice descriptor is at byte offset 2 x 3 = 6 and is byte reversed.

c) Byte offset 1 of the slice gives the index into the SAT for the PCB memory address (Table 6.3.1).

d) The COP Table is located in the PCB at the memory address held in bytes offset 10 and 11 (reversed) (Table 8.2.1).

6.4 OVERLAYABLE AREA AND SEGMENTS

Above the locked memory area is the overlayable area. PTRX of MAP VMWA (Table 7.5.1) addresses the first byte and PTRZ the last byte of this area. To obtain an analysis of the structure of this area use the PRINT MEMORY.LINKS option of PMB80.

The overlayable area contains code and data segments and may also contain available memory areas. All segments are accessed through the slice structure and Data Segment Tables as explained above. Available areas are also treated as segments and have a segment descriptor (with flags = @00@) in the first eight bytes of the area.

The segments in the overlayable area are linked together in the following manner (see figure 6.4.1). The first two bytes addressed by PTRX provide the memory address of the segment descriptor for the first segment in the overlayable area.

This segment descriptor contains the absolute memory address and length of its segment. Immediately following this segment in memory, the next two bytes hold the memory address of the segment descriptor of the second segment in the overlayable area. This is repeated until the memory addressed by PTRZ is reached.

If an available area is shorter than ten bytes in length (two for the memory link and eight for the segment descriptor), then it is filled with binary zeros. A link with the first byte @00@ is invalid because this would point to ROM, and the virtual memory algorithms recognise that this byte is a filler, not a memory link .

6.5 PHT AREA

The Peripheral Handling Table (PHT) area is located at the top of page zero of memory. The contents are analysed by the PRINT PHT option of PMB80. The area contains tables of information used by the Master Interrupt Processor (MIP) and DDR code. Each table is loaded into memory at warmstart time and remains at the same memory address throughout the session. The Tables are addressed by the field PHT.ADDR.TABLE of MAP PHDMP (Table 7.2.1). The contents of these tables are discussed in more detail in section 9.

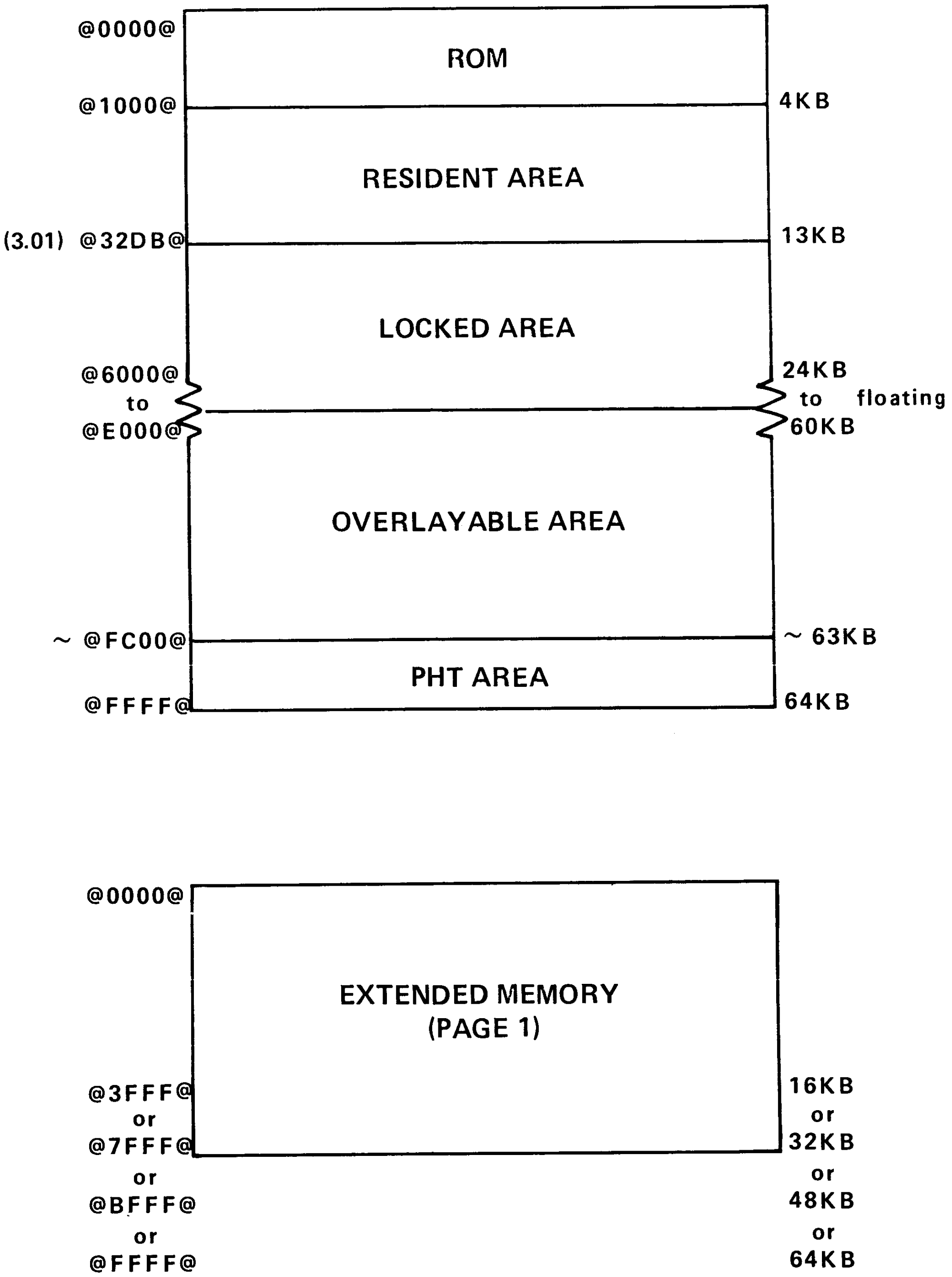
6.6 EXTENDED MEMORY

Extended (Page 1) memory may contain only a few selected items. To determine which items are in extended memory, use the PRINT MEMORY.MAP option of PMB80.

Any or all of the interpreter segments, the index communicate handler segment, or the data comm message buffer segment may be held in this memory. If a slice is located in extended memory, then all segments of that slice are included in the extended memory.

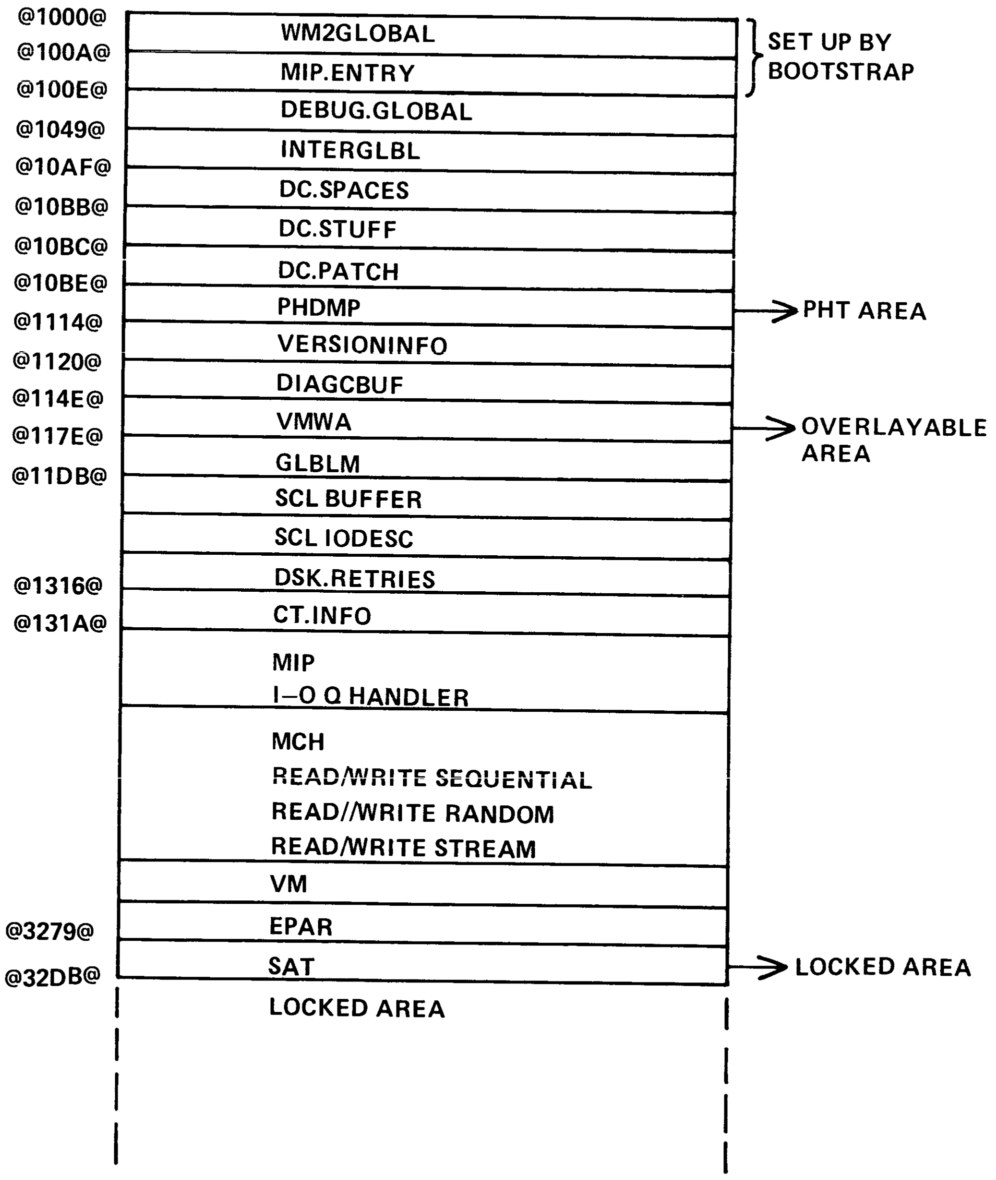
The segments (excluding data comm buffers) are loaded at warmstart time as specified by the SYSCONFIG file and subject to there being sufficient extended memory space. These segments then remain at their allocated memory locations for the entire session.

6.1.1 AREAS OF MEMORY



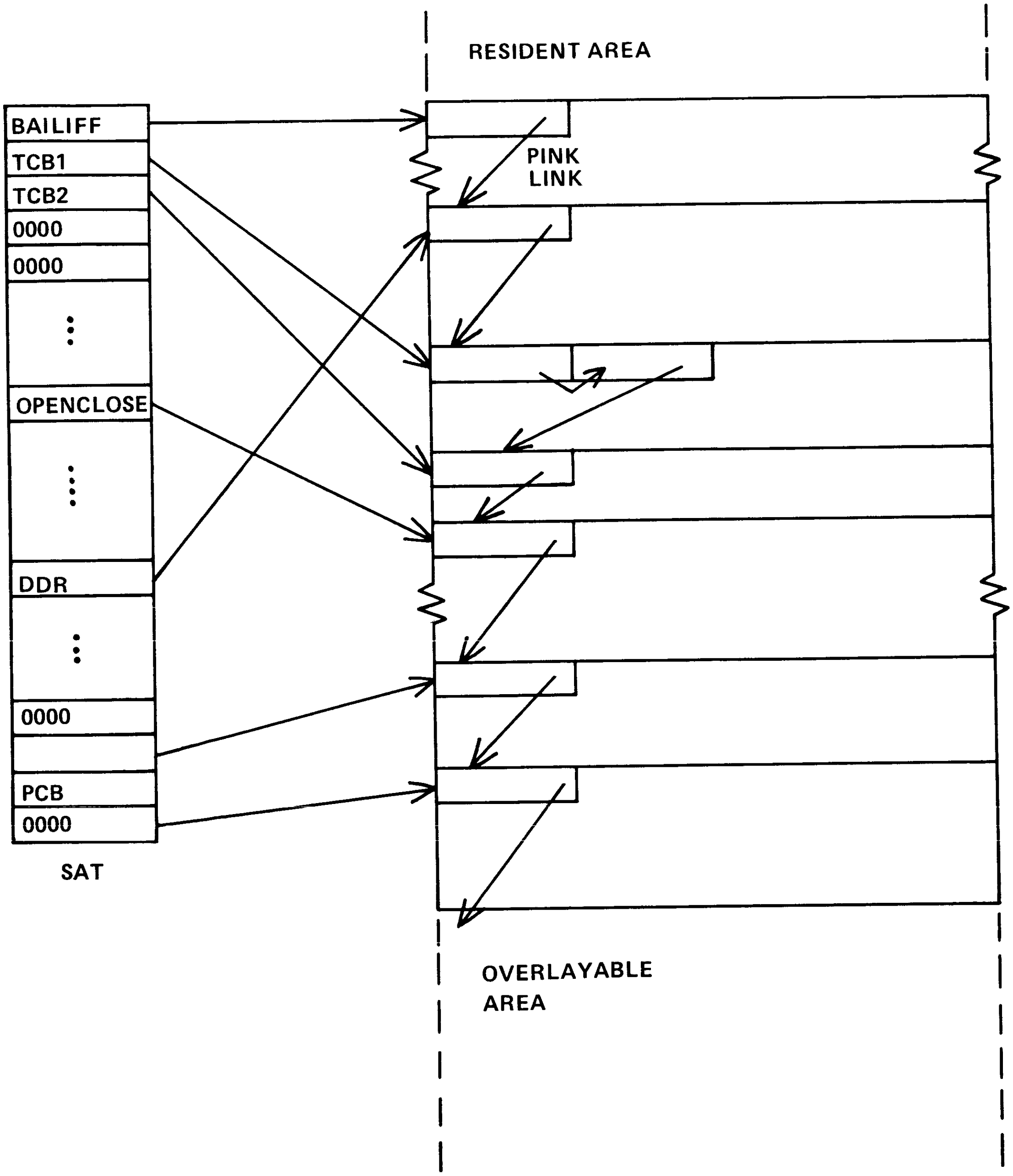
AREAS OF MEMORY

6.2.1 RESIDENT AREA LAYOUT



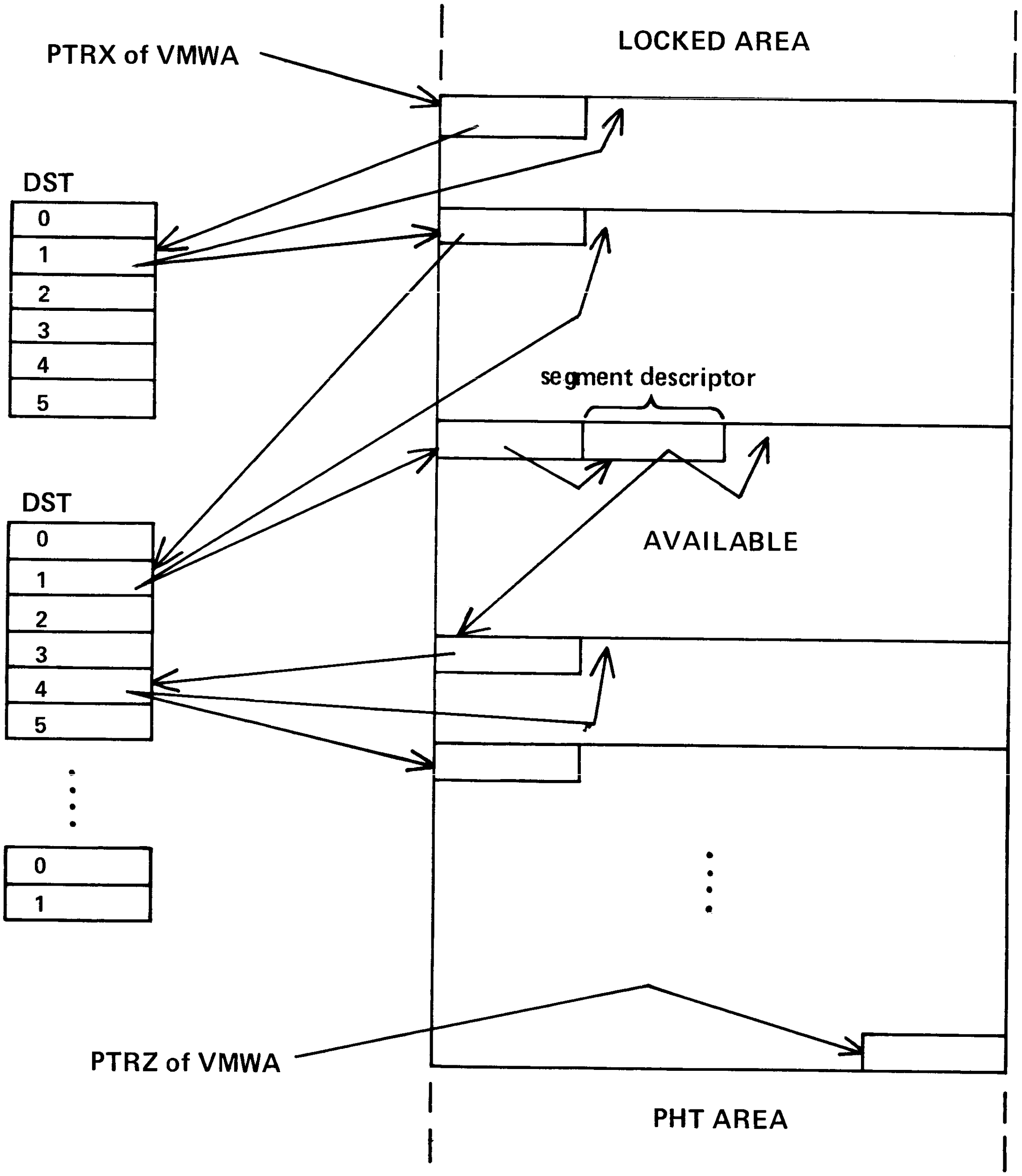
RESIDENT AREA LAYOUT

6.3.1 LOCKED AREA LAYOUT



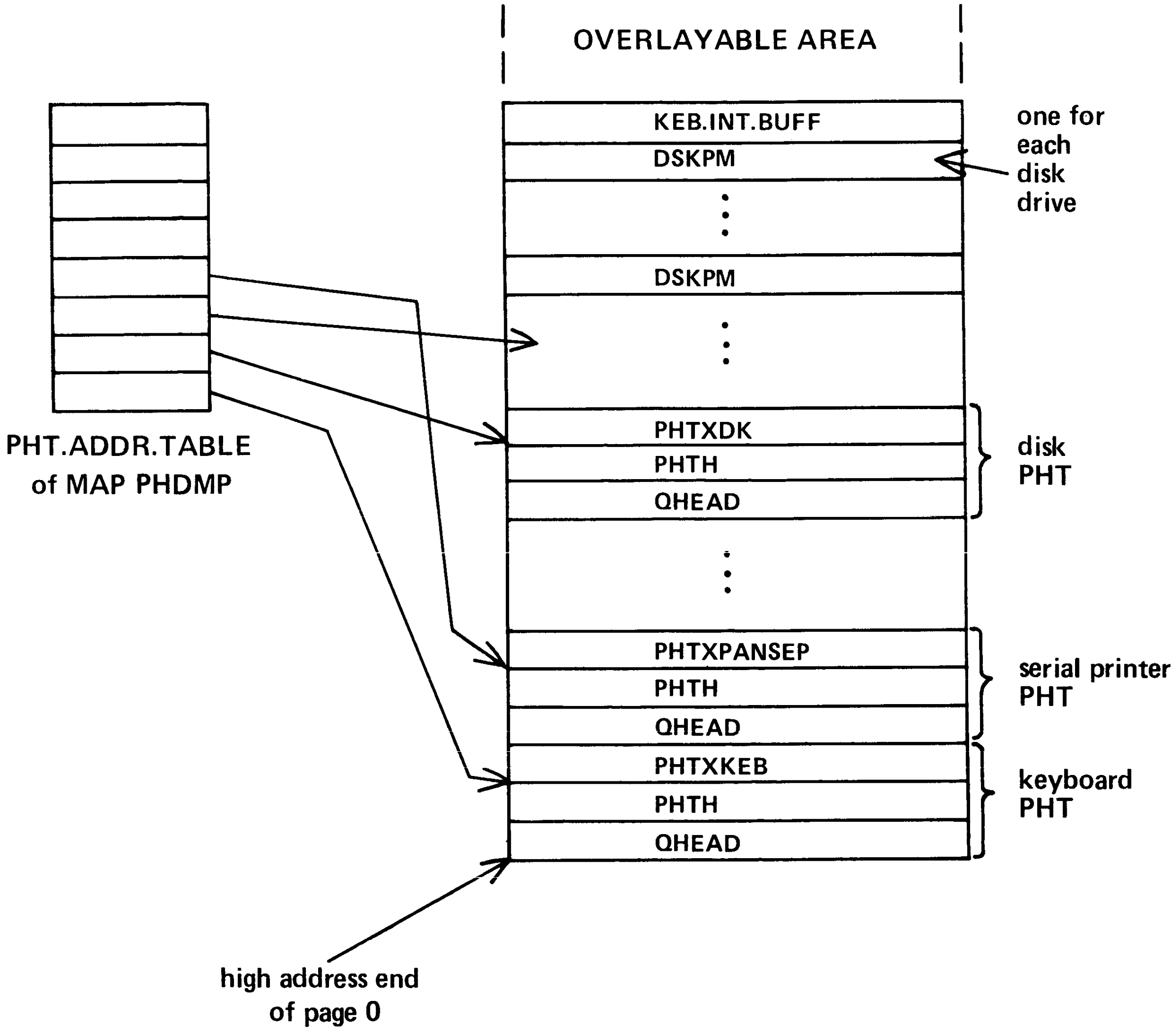
LOCKED AREA LAYOUT

6.4.1 OVERLAYABLE AREA LAYOUT



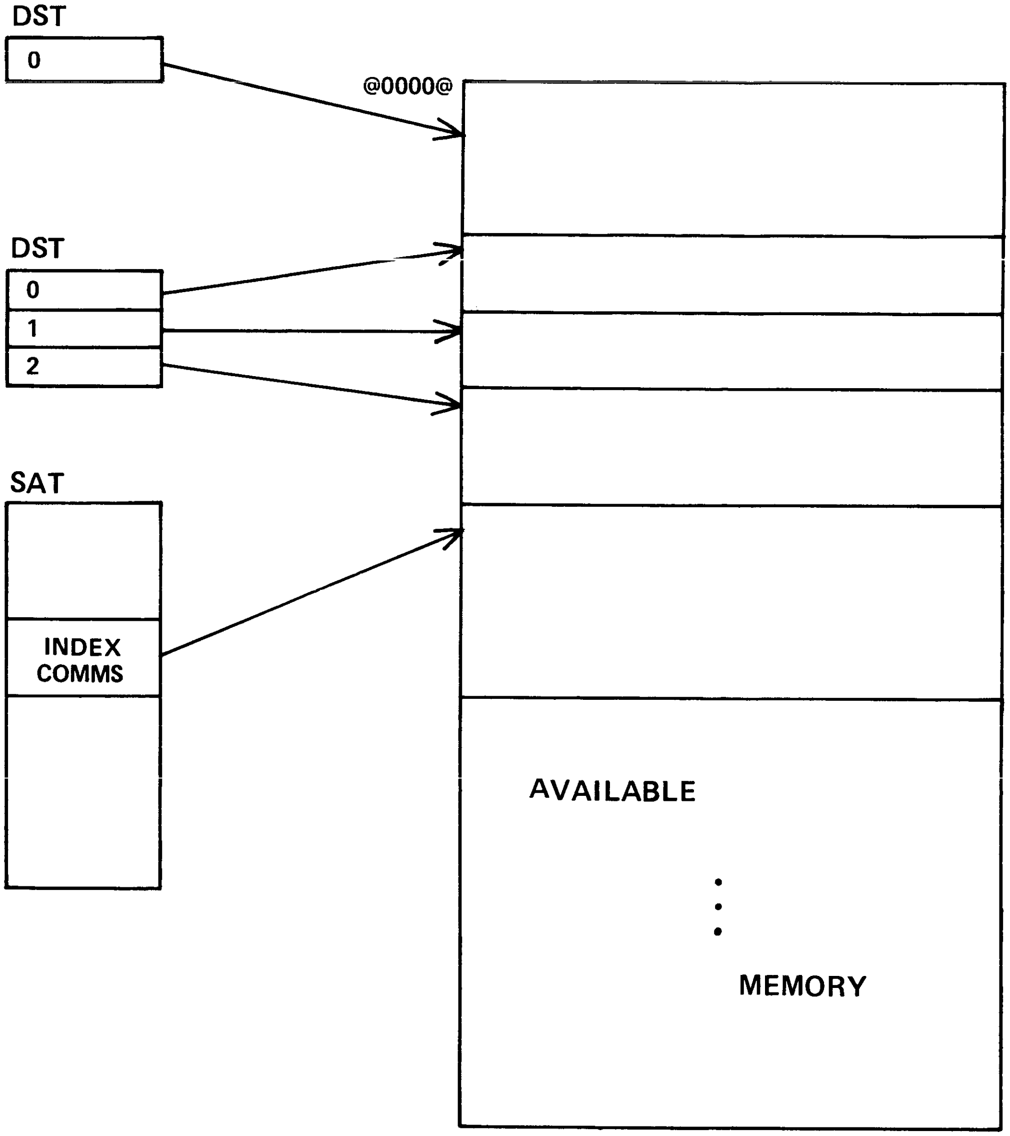
OVERLAYABLE AREA LAYOUT

6.5.1 PHT AREA LAYOUT

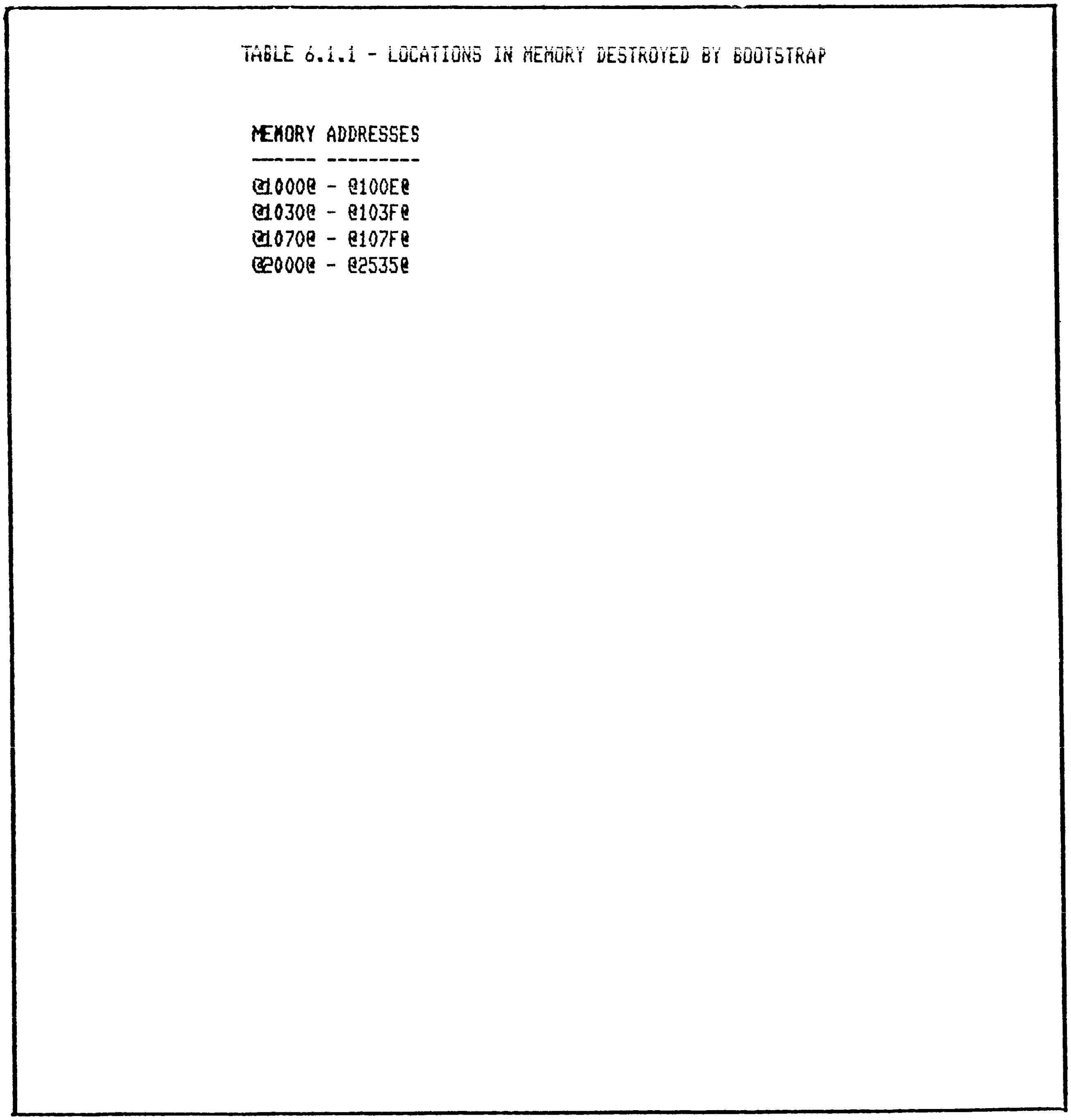


PHT AREA LAYOUT

6.6.1 EXTENDED MEMORY LAYOUT



EXTENDED MEMORY LAYOUT



```
TABLE 6.1.1 - LOCATIONS IN MEMORY DESTROYED BY BOOTSTRAP


MEMORY ADDRESSES
------ ---------
@1000@ - @100E@
@1030@ - @103F@
@1070@ - @107F@
@2000@ - @2535@
```

LOCATIONS IN MEMORY DESTROYED BY BOOTSTRAP

TABLE 6.2.1 - SATM MAP (3.01) - SLICE ADDRESS TABLE

OFF	LEN			
SET	GTH	FIELD	USE	FORMAT
0	-	OSRCSN	memory addresses of slice descriptors	LABEL
0	2	BLFSN	BAILIFF Task Control Block (TCB) addess	AA BR
2	16	TSK1SN	addresses for mix numbers 1-8 (TCBs)	"
18	6	AVRSN	Automatic Volume Recognition (AVR) TCB address	"
			and utility mix numbers 10-11 TCB addresses	-
24	2	SUSN	SYS-SUPERUTL (mix=12) TCB address	"
26	2	MCSSN	Message Control System (MCS mix=13) TCB address	"
28	2	NDLSN	Network control program (mix=14) TCB address	"
30	2	LDRSN	program LOADER (mix=15) TCB address	"
32	2	SNABS	absolute memory address slot	"
			used for entry to global MCP code	-
34	2	DKADPSLC	-	=
"	"	OPCLSN	OPEN-CLOSE-AVR code slice address	"
36	2	INITSN	WARMSTART-INITIALISE code slice address	"
38	2	DISPSN	DISPLAY-SPO code slice address	"
40	2	LPRSN	LINE PRINTER DDR code slice address	"
42	2	CSTRSN	CASSETTE DDR code slice address	"
44	2	DSKRSN	DISK DDR code slice address	"
46	2	SEPRSN	SENNEFFE (60 cps) serial printer DDR address	"
48	2	KEBRSN	KEYBOARD DDR address	"
50	2	SCREENSN	SELF SCAN(SS1 or SS2) and CRT DDR code slice address	"
52	2	ADCRSN	ASYNC DATA COMM DDR code slice address	"
54	2	SDCRSN	SYNC DATA COMM DDR code slice address	"
56	2	DCSLICE	DATA COMM COMMUNICATE handling code slice address	"
58	2	CONSN	CONSOLE COMMUNICATE handling code slice address	"
60	2	PANRSN	PANTIN (120 and 180cps) serial printer DDR address	"
62	2	INXSN	INDEXED FILE COMMUNICATE handling code slice address	"
64	2	RTCRSN	REAL TIME CLOCK DDR code slice address	"
66	2	ICMSRSN	ICMD DDR code slice address	"
68	2	MESSSN	DATA COMM MESSAGE BUFFER SPACE slice address	"
70	2	CONBUFSN	CONSOLE FILE BUFFER (non SCL keyboard input) address	"
72	2	SCLBUFSN	SCL (keyboard and ZIP) BUFFER slice address	"
74	2	DIAGSN	TRACE DIAGNOSTICS MCP code slice address	"
76	2	OCOM.SN	OVERLAYABLE COMMUNICATE handling code slice address	"
78	20	FREESN	two-byte address slots used as required for	"
			Program Control Block (PCB) program code slice addresses and	-
			Interpreter Control Block (ICB) interpreter addresses	-
			slices 39-48	-

SATM MAP - SLICE ADDRESS TABLE

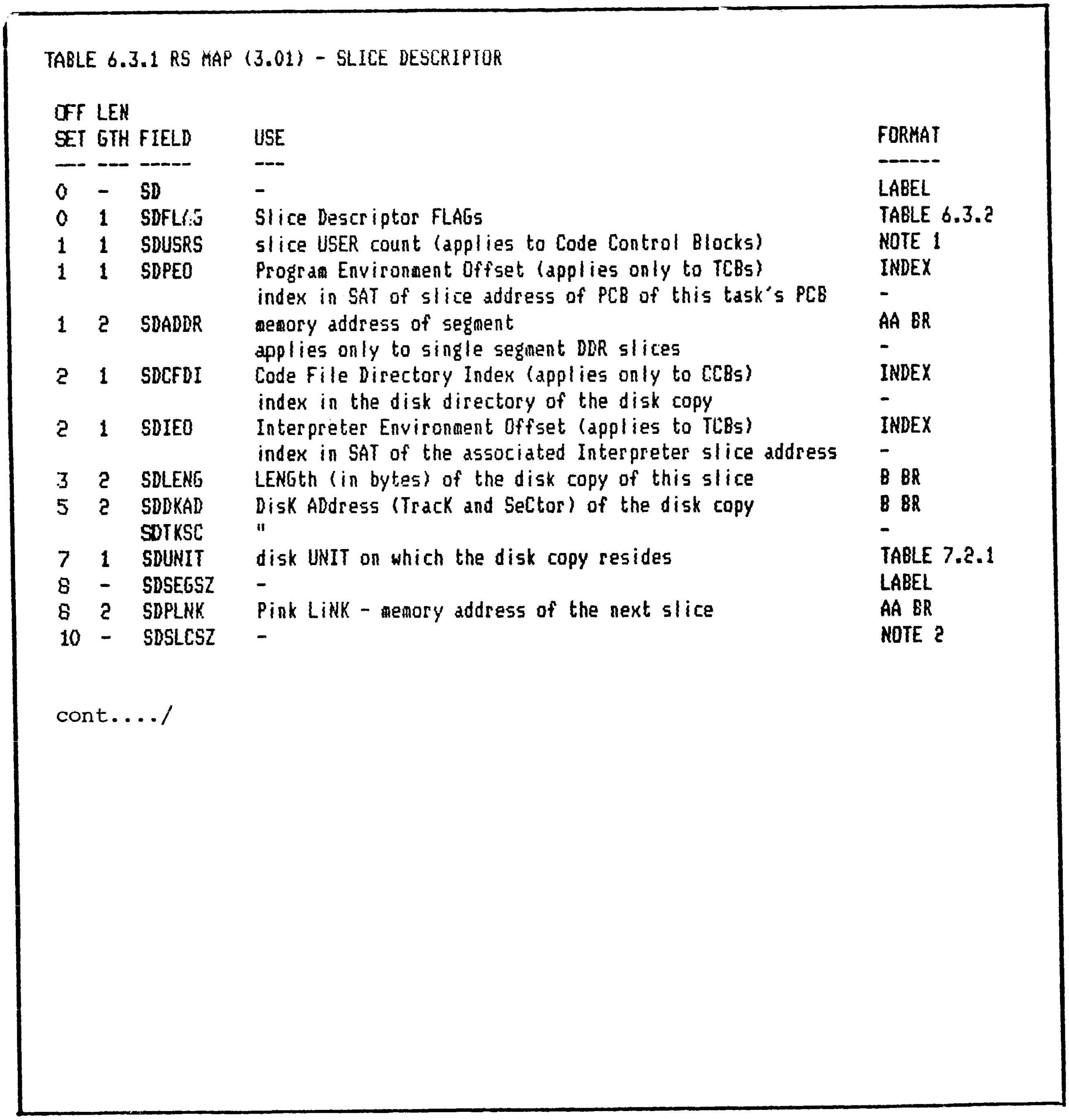


TABLE 6.3.1 RS MAP (3.01) - SLICE DESCRIPTOR

OFF SET	LEN GTH	FIELD	USE	FORMAT
0	-	SD	-	LABEL
0	1	SDFLAG	Slice Descriptor FLAGs	TABLE 6.3.2
1	1	SDUSRS	slice USER count (applies to Code Control Blocks)	NOTE 1
1	1	SDPEO	Program Environment Offset (applies only to TCBs)	INDEX
			index in SAT of slice address of PCB of this task's PCB	-
1	2	SDADDR	memory address of segment	AA BR
			applies only to single segment DDR slices	-
2	1	SDCFDI	Code File Directory Index (applies only to CCBs)	INDEX
			index in the disk directory of the disk copy	-
2	1	SDIEO	Interpreter Environment Offset (applies to TCBs)	INDEX
			index in SAT of the associated Interpreter slice address	-
3	2	SDLENG	LENGth (in bytes) of the disk copy of this slice	B BR
5	2	SDDKAD	DisK ADdress (TracK and SeCtor) of the disk copy	B BR
		SDTKSC	"	-
7	1	SDUNIT	disk UNIT on which the disk copy resides	TABLE 7.2.1
8	-	SDSEGSZ	-	LABEL
8	2	SDPLNK	Pink LiNK - memory address of the next slice	AA BR
10	-	SDSLCSZ	-	NOTE 2

cont..../

RS MAP (3.01) - SLICE DESCRIPTOR

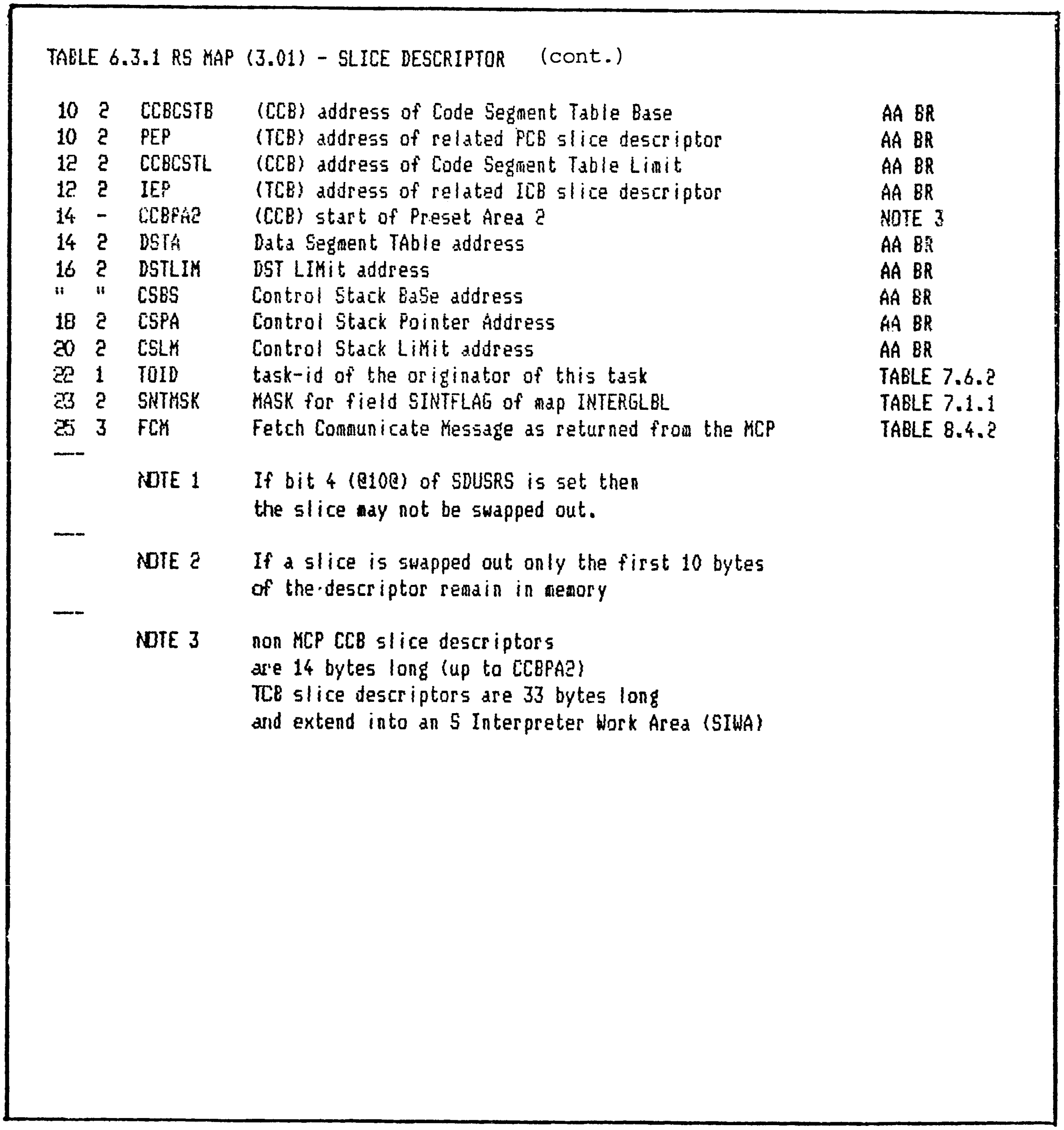


TABLE 6.3.1 RS MAP (3.01) - SLICE DESCRIPTOR (cont.)

10	2	CCBCSTB	(CCB) address of Code Segment Table Base	AA BR
10	2	PEP	(TCB) address of related PCB slice descriptor	AA BR
12	2	CCBCSTL	(CCB) address of Code Segment Table Limit	AA BR
12	2	IEP	(TCB) address of related ICB slice descriptor	AA BR
14	-	CCBPA2	(CCB) start of Preset Area 2	NOTE 3
14	2	DSTA	Data Segment TAble address	AA BR
16	2	DSTLIM	DST LIMit address	AA BR
"	"	CSBS	Control Stack BaSe address	AA BR
18	2	CSPA	Control Stack Pointer Address	AA BR
20	2	CSLM	Control Stack LiMit address	AA BR
22	1	TOID	task-id of the originator of this task	TABLE 7.6.2
23	2	SNTMSK	MASK for field SINTFLAG of map INTERGLBL	TABLE 7.1.1
25	3	FCM	Fetch Communicate Message as returned from the MCP	TABLE 8.4.2

NOTE 1 If bit 4 (@10@) of SDUSRS is set then
the slice may not be swapped out.

NOTE 2 If a slice is swapped out only the first 10 bytes
of the descriptor remain in memory

NOTE 3 non MCP CCB slice descriptors
are 14 bytes long (up to CCBPA2)
TCB slice descriptors are 33 bytes long
and extend into an S Interpreter Work Area (SIWA)

RS MAP (3.01) - SLICE DESCRIPTOR (cont.)

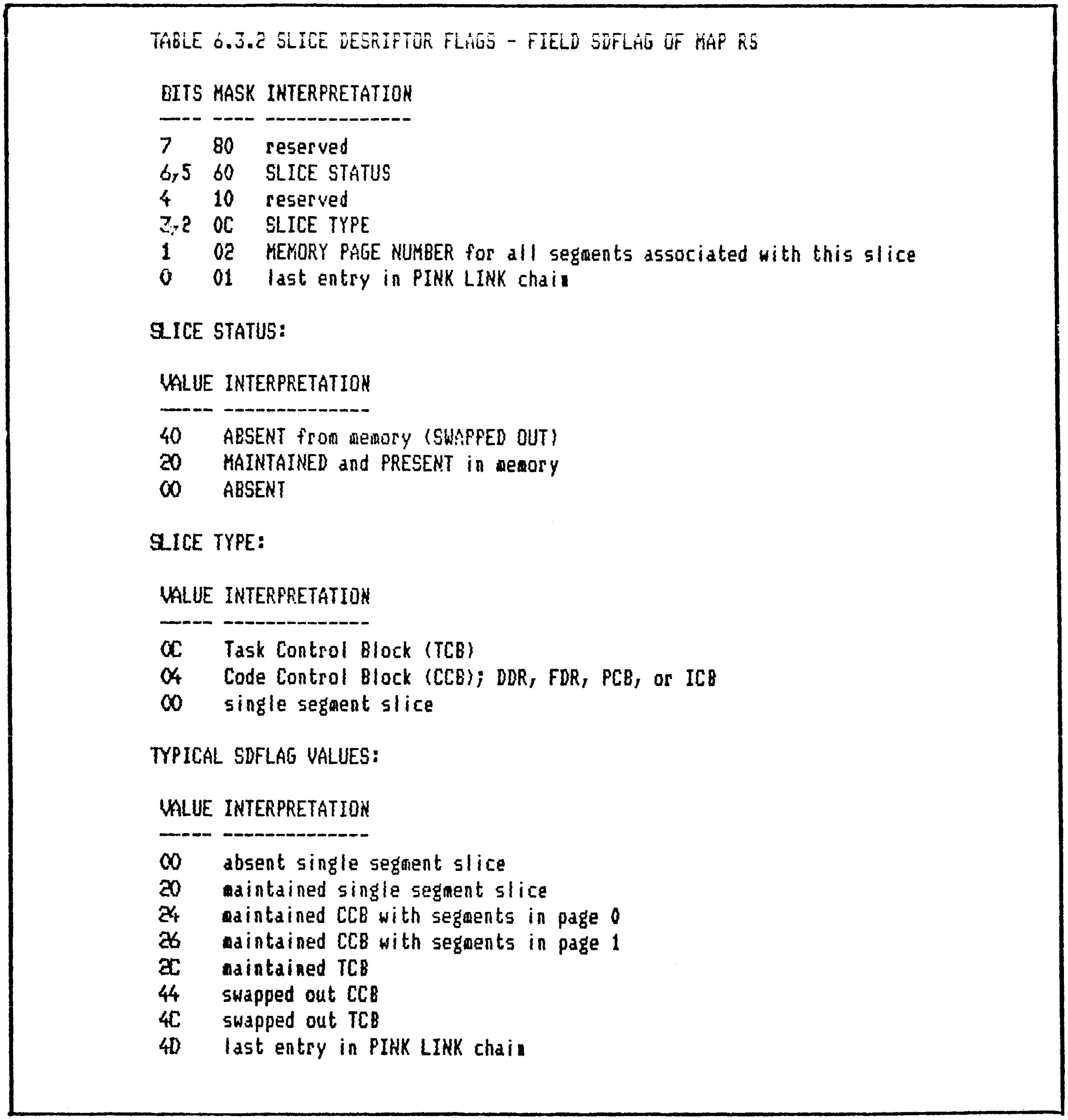


TABLE 6.3.2 SLICE DESRIPTOR FLAGS - FIELD SDFLAG OF MAP RS

BITS	MASK	INTERPRETATION
7	80	reserved
6,5	60	SLICE STATUS
4	10	reserved
3,2	0C	SLICE TYPE
1	02	MEMORY PAGE NUMBER for all segments associated with this slice
0	01	last entry in PINK LINK chain

SLICE STATUS:

VALUE	INTERPRETATION
40	ABSENT from memory (SWAPPED OUT)
20	MAINTAINED and PRESENT in memory
00	ABSENT

SLICE TYPE:

VALUE	INTERPRETATION
0C	Task Control Block (TCB)
04	Code Control Block (CCB); DDR, FDR, PCB, or ICB
00	single segment slice

TYPICAL SDFLAG VALUES:

VALUE	INTERPRETATION
00	absent single segment slice
20	maintained single segment slice
24	maintained CCB with segments in page 0
26	maintained CCB with segments in page 1
2C	maintained TCB
44	swapped out CCB
4C	swapped out TCB
4D	last entry in PINK LINK chain

SLICE DESCRIPTOR FLAGS - FIELD SDFLAG OF MAP RS

TABLE 6.4.1 - SEGD MAP - SEGMENT DESCRIPTOR

OFF	LEN			
SET	GTH	FIELD	USE	FORMAT
0	1	SGDFL	SeGment Descriptor FLags	TABLE 6.4.2
1	2	SGDSS	Segment Start address in memory	AA BR
3	2	SGDSL	number of bytes in the segment	B BR
5	2	SGDDA	Disk Address (zero relative sector) of the disk copy	B BR
7	1	SGDDU	Disk Unit holding the disk copy	TABLE 7.2.1
8	-	SZ.SEG.DESC	-	LABEL

TABLE 6.4.2 - SEGMENT DESCRIPTOR FLAGS - FIELD SGDFL OF MAP SEGD

BITS	MASK	INTERPRETATION
7	80	1 = segment USED recently
6	40	1 = segment ABSENT from memory (overlayed)
5	20	1 = segment OVERLAYABLE (not locked)
4	10	reserved
3	08	1 = segment may be updated (READ-WRITE)
2	04	1 = segment is not for system use
1	02	reserved
0	01	1 = segment has been updated since it was read into memory

TYPICAL VALUES FOR MOST SIGNIFICANT DIGIT:

VALUE	INTERPRETATION
A, 2	OVERLAYABLE segment PRESENT in memory
8, 0	LOCKED segment (will be located in slice)
6, 4	ABSENT (overlayed) segment

TYPICAL VALUES FOR LEAST SIGNIFICANT DIGIT:

VALUE	INTERPRETATION
D, 9	segment UPDATED
C,8,4	segment NOT UPDATED

SEGD MAP - SEGMENT DESCRIPTOR and
SEGMENT DESCRIPTOR FLAGS - FIELD SGDFL OF MAP SEGD

SECTION 7

7. GLOBAL MCP MAPS AND TABLES

This section discusses the information held in the Global MCP maps and tables. These maps should be examined in detail if the preliminary analysis discussed in section four is insufficient.

On each release of the MCP, these tables are located at a specific address in memory. As the offset and format of each map may change between release levels, PMB80 handles this by having different reference files PMBM.nnnnn and PMBO.nnnnn (see section 3).

The global maps are analysed by PMB80 using the PRINT GLOBAL option. Single maps may be selected for printing by the appropriate parameter to the print command (GWA, DIAGNOSTICS, VMWA, ESCT or SCL).

7.1 MCP - INTERPRETER INTERFACE

The interpreter global work area (Table 7.1.1) holds the memory addresses of various global MCP routines. It also holds the first link into the memory organisation through SATLINK. A check should be made that these fields hold valid values. For example, the Global MCP routines lie below the locked slice area between @1000@ and @4000@, so the addresses of the global routines should be in this range

Another interesting field is TOTSICT. This is used in thrashing detection which is discussed in section 7.5.3.

7.2 PERIPHERAL INTERRUPT HANDLING

The peripheral handling dump area (PHDMP MAP Table 7.2.1) is used when the MCP is handling hardware interrupts, and when printing trace diagnostics.

When a hard interrupt is received from an I-O channel, the ROM code causes entry to the Master Interrupt Processor (MIP). This global MCP code stores the contents of the processor registers in PHDMP, and performs basic interrupt handling.

7.2 PERIPHERAL INTERRUPT HANDLING (CONT.)

If the General Trace (GT) facility is switched on (see section 5) the processor registers are stored in PHDMP each time a trace point is encountered. If the system fails and issues a diagnostic on PK-lights 17-24 (see section 5) the processor registers are also stored in PHDMP.

Thus, the register dump fields of PHDMP contain the contents of the processor registers, either at the last I-O interrupt or the last diagnostic trace point. The latter case can be distinguished since PHDMPAD will equal the last one byte diagnostic in DIAGCIRC (Table 7.4.1).

In either case, the contents of these dump areas usually give the most accurate information on which function the MCP was performing before the dump was taken. The interpretation of the registers is explained in section 5.1.

The remaining fields of PHDMP are used by the MIP in handling I-O interrupts. The field INTMASK gives the I-O channels on which interrupts may be processed. The field INT.SCAN.TABLE provides the order in which interrupts should be processed. The field ISCT gives the channel which was last serviced; this field points to the PHT associated with that channel.

7.3 KEYBOARD VERSION AND SPO TYPE

The VERSIONINFO MAP (Table 7.3.1) indicates which translation table the MCP is using for printing characters.

If wrong characters are being printed (£,#, $, etc.) this field should be checked and an engineer should be asked to give advice about correct installation of the machine backplane.

7.4 DIAGNOSTIC INFORMATION

As its name suggests, the DIAGCBUF MAP (Table 7.4.1) holds information specifically aimed at assisting problem diagnosis. It consists of three parts. The circular buffer of one byte diagnostics (DIAGCIRC) indicates what the MCP has been doing; that is, which of its global and slice code has been running. The register save areas give the task which is running and which microcode this task is using.

The light switch indicates whether the keyboard was enabled at the time that the dump was taken.

The PRINT DIAGNOSTICS option of PMB80 causes this map to be printed.

7.4.1 ONE BYTE DIAGNOSTICS

Trace diagnostics are discussed in section 5. Each time a trace point is encountered, the one-byte trace diagnostic is inserted in DIAGCBUF. This is done whether or not the MCP trace function has been switched on using the GT command.

Field DIAGNINDEX of DIAGCBUF is the byte offset (zero relative) of the position of the next entry to be made in DIAGCIRC. With the entries in DIAGCIRC and reference to tables 5.3.1 to 5.3.11, it is usually possible to determine what the MCP was doing immediately before the dump was taken.

No entries are made in DIAGCIRC when peripheral interrupts occur. This should always be remembered, and reference made to the PHDMP map to determine when the last interrupt occurred (see section 7.2)

7.4.2 TOP OF CONTROL STACK

The switching of control between tasks is performed by the EPAR (section 7.6.1). 'When EPAR exits and control is passed to a task, a diagnostic with value @EE@ is entered in DIAGCIRC and the registers XY, L and J are saved in XYSAVE, LSAVE and JSAVE. The contents of these registers at this time indicates which task gained control and which microcode was entered from EPAR.

XYSAVE holds a copy of the top four bytes of the control stack of the task which was entered, (see table 7.4.2), and indicates in which microcode segment and at what offset the entry occurred. LSAVE holds the memory address (byte reversed) of the task slice descriptor. JSAVE holds the memory address (byte reversed) of the microcode segment to be executed.

Only microcode may execute on the B80 processor; therefore all tasks except microcoded tasks need an interpreter to be executed. Even microcoded tasks make heavy use of the MCP routines for peripheral and file handling. Thus it is very rare to find the entry code in XYSAVE referring to a task slice. The entry code usually refers to one of the following (refer to Table 7.4.2):

7. GLOBAL MCP MAPS AND TABLES (CONT.)

7.4 DIAGNOSTIC INFORMATION (CONT.)

7.4.2 TOP OF CONTROL STACK (CONT.)

INTERPRETER : It is very unusual for problems to be encountered in an interpreter. The problem is likely to be related to a hardware interrupt.

DDR : Many program communicates require input-output operations. These will be performed by an MCP DDR slice.

MCP FUNCTION : Some program communicates require the execution of a Function Dependent Routine of the MCP. For example, file open and close, index file communicates, console file communicates.

ABSOLUTE ADDRESS : Global MCP routines are entered at an absolute address. These addresses are stored in the INTERGLBL map (see table 7.1.1, and section 7.1).

BAILIFF (slice 0) - a microcoded MCP task, see Section7.6.3
SCL/LOADER (slice 15) - a microcoded MCP task, see Section 7.7.

NOTE:

Slice numbers in XYSAVE are twice the value expected. They provide an index into SAT which has entries two bytes long. For example, if XYSAVE has a value @12220000@, then the last code entered was slice 17 (=@22@/2), segment 18 offset 0, a segment of the MCP OPEN-CLOSE slice.

7.5 VIRTUAL MEMORY SUBSYSTEM

All movement of segments in and out of memory is the responsibility of the Virtual Memory (VM) subsystem. The VMWA (Table 7.5.1) is its work area.

The work area contains memory addresses, lengths, an I-O descriptor for loading and unloading segments, and a "GETSEG" counter.

7.5.1 MEMORY LINKS

Without refernce to the MCP listing it is difficult to be certain of errors in the VM pointers. If a virtual memory operation is in process, the pointers may be in a transient state and be marked on the dump as "illegal". (The PRINT MEMORY.LINKS option of PMB80 may produce spurious errors on such a occasion).

If a memory link becomes overwritten by some other part of the MCP, then the system may continue to run satisfactorily until the VM subsystem encounters the corruption. For this reason it is possible for the VM routine to fail due to a fault in another part of the system. A dump showing a fault in the memory links may not give the solution to a problem but rather the symptoms of a completely independent fault.

7.5.2 VIRTUAL MEMORY I-O

The virtual memory I-O has the standard I-O descriptor format (see table 9.3.3). The most important field is the flags IODFL.

Since, in a virtual memory system, disk is an extension of main memory, a disk "parity error" has very serious implications. If the B80 MCP is unable to access any of its virtual memory on disk, then execution is halted and a "DF" diagnostic is displayed. The virtual memory I-O descriptor contains further information about the failure. The flags define the nature of the error (see table 9.3.4), the disk unit and address fields locate the area of failure.

7.5.3 THRASHING DETECTION

The virtual memory subsystem is responsible for thrashing detection. It uses the field GETCNTR, and the field TOTSICT of map INTERGLBL (table 7.1.1).
GETCNTR is set to @100@ and decremented by one each time a segment is read from disk into memory. At the same time, TOTSICT is set to zero and incremented by one for each S-OP processed by an interpreter. GETCNTR provides a measure of the "wasted" work and TOTSICT measures the "useful" work. If the ratio (256-GETCNTR): TOTSICT becomes greater than 1:100, then the MCP takes action to reduce thrashing. As the fields are periodically reset, no useful information can be derived if GETCNTR is close to @100@ and TOTSICT is close to 0.

7.6 SYSTEM CONTROL

The MCP has three resources to manage; the processor, the MCP routines used by tasks, and the memory. The processor and MCP functions are allocated by the Execution Priority Assignment Routine (EPAR), the task control subsection. The memory is allocated by the Bailiff,task 0.

The global MCP map GLBLM (Table 7.6.1) is the work area used by these sections of the MCP to control the allocation of these resources. The PRINT GLBLM option of PMB80 prints the contents of this map. The PRINT MIX option provides an analysis of some of the more useful fields of this map.

7.6.1 PROCESSOR ALLOCATION - EPAR

The EPAR manages the control of the processor by a system of "independent runners". These independent runners are tasks, each having a task-id and a corresponding mix number between 0 and 15 (see table 7.6.2).

At any time, only one of these tasks is the current task and has control of the processor.

Each task has an entry in the Execution Scan Table (ESCT). The field EICT of map GLBLM points to the entry in ESCT for the CURRENT task. This byte, also called the task-id is displayed on the keyboard D-lights when control is passed to the task.

Section 7.4.2 describes how to determine the function that the task is currently performing.

The status of all the tasks in the system is stored in the Wait Key Table (WAKT). Each entry in the ESCT has a one byte entry in the WAKT in the corresponding position. The order in which the task-ids occur in the ESCT and WAKT reflects the relative priorities of the tasks.

The highest priority tasks occur first. The first six tasks in the table have specific functions and always occur first, followed by "user" tasks.

The value of the entry in the WAKT indicates whether the task is runnable or waiting, plus the cause of waiting if applicable (see Table 7.6.3).

7.6.2 MCP FUNCTION CONTROL - LOCKS

Some MCP routines may not be used by two tasks simultaneously. Each of these routines has a lock. These locks reside in the GLBLM map (table 7.6.1). If a task is using one of these routines (for example: opening a file, or displaying a message on the SPO) then the lock corresponding to that routine is set to the task-id. Such a lock and its corresponding MCP routine may not then be used by another task, which will have to wait.

There may exist problems with the use of locks which could cause either a program or the complete system to hang. If there are two tasks holding different locks, and both of them are waiting for the lock which the other task holds, then a deadlock situation arises. Neither task can proceed.

7.6.3 MEMORY ALLOCATION - BAILIFF

The Bailiff (task 0) controls the allocation of memory to tasks. If there is sufficient memory for all tasks in the mix, then the virtual memory subsystem will be overlaying segments in the overlayable area as required. If however, there is a shortage of memory, then thrashing will be detected and the Bailiff will come into operation.

The Bailiff swaps slices in and out of memory. Swapping should not be confused with the segment overlaying operation performed by the virtual memory subsystem. Swapping is an operation performed on a locked slice. When a slice is swapped out, the entire locked slice (excluding the first ten bytes of the slice descriptor) is written out to disk and the other contents of the locked slice area are moved down in memory. In this way, more overlayable memory becomes available.

Slices may volunteer for eviction by having no users, or being long waited, or they may be conscripted as required.

Volunteers will only be swapped out when either enough volunteers exist to make it worthwhile (field SWAPCNT of map GLBLM is small enough), or thrashing has been detected. If thrashing is detected and no volunteers exist, then a slice (usually a TCB) will be conscripted. If a task (TCB) is swapped out then it is no longer runnable. The task will be restored when more memory becomes available.

If a program is written which has very large segments (either data or code) then it is possible for thrashing to be detected when this is the only program running.

In this case, the task will be evicted and never restored since no more memory becomes available. The operator must intervene with a GO command.

The work areas used by the Bailiff to make decisions on swapping can be found in the GLBLM map (table 7.6.1) with further explanation in table 7.6.4.

7.7 SCL INTERPRETER AND LOADER

Task number 15 is reserved for the MCP slice which interprets the System Control Language (SCL) commands, and loads programs. When this task is running, the D-lights D4, D5, D6, D7 will all be on.

The SCLBUFFER map (Table 7.7.1) holds the last SCL input (either from the keyboard or from a ZIP) and is terminated by @1F@.

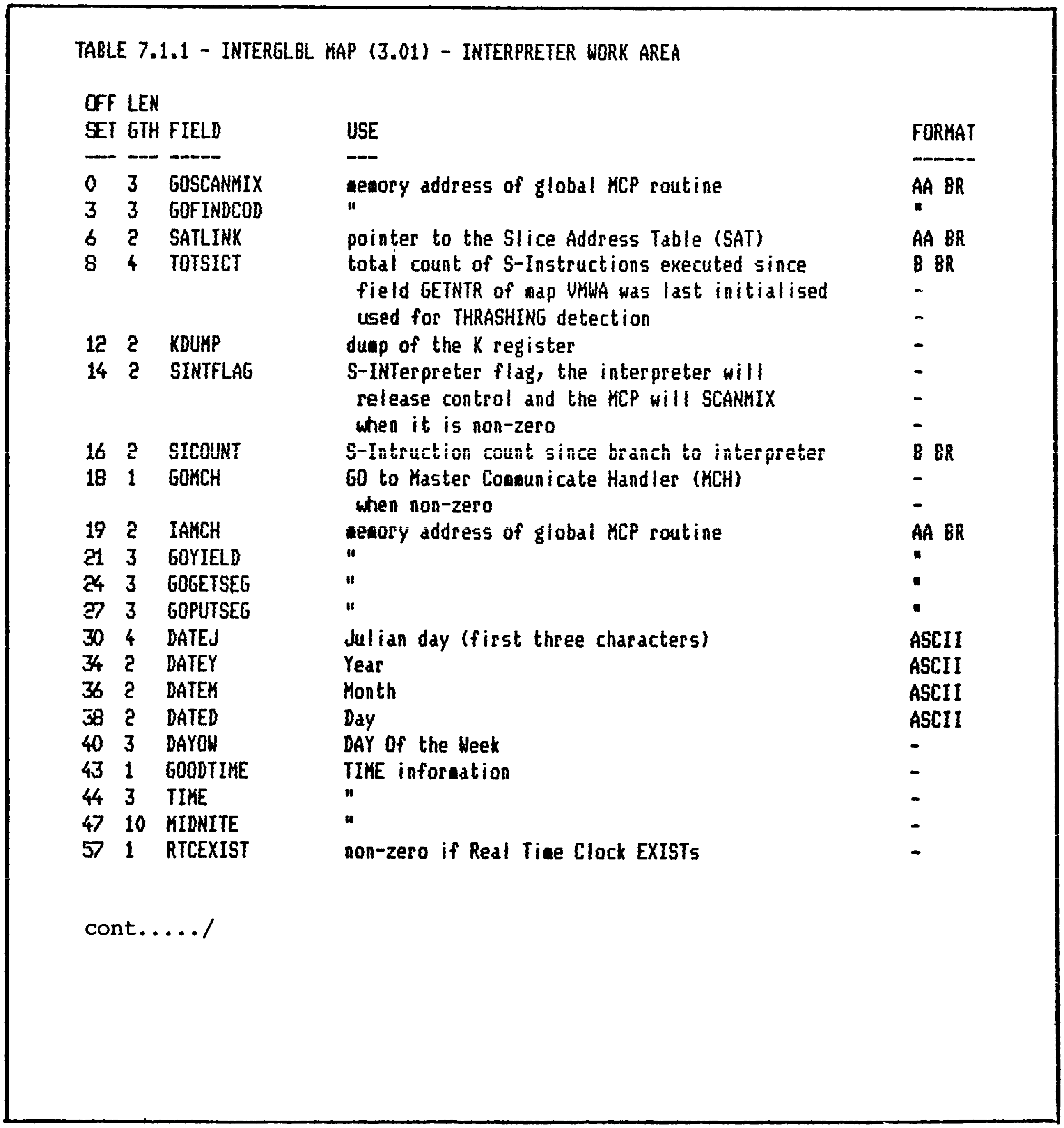


TABLE 7.1.1 - INTERGLBL MAP (3.01) - INTERPRETER WORK AREA

OFF	LEN			
SET	GTH	FIELD	USE	FORMAT
0	3	GOSCANMIX	memory address of global MCP routine	AA BR
3	3	GOFINDCOD	"	"
6	2	SATLINK	pointer to the Slice Address Table (SAT)	AA BR
8	4	TOTSICT	total count of S-Instructions executed since	B BR
			field GETNTR of map VMWA was last initialised	-
			used for THRASHING detection	-
12	2	KDUMP	dump of the K register	-
14	2	SINTFLAG	S-INTerpreter flag, the interpreter will	-
			release control and the MCP will SCANMIX	-
			when it is non-zero	-
16	2	SICOUNT	S-Intruction count since branch to interpreter	B BR
18	1	GOMCH	GO to Master Communicate Handler (MCH)	-
			when non-zero	-
19	2	IAMCH	memory address of global MCP routine	AA BR
21	3	GOYIELD	"	"
24	3	GOGETSEG	"	"
27	3	GOPUTSEG	"	"
30	4	DATEJ	Julian day (first three characters)	ASCII
34	2	DATEY	Year	ASCII
36	2	DATEM	Month	ASCII
38	2	DATED	Day	ASCII
40	3	DAYOW	DAY Of the Week	-
43	1	GOODTIME	TIME information	-
44	3	TIME	"	-
47	10	MIDNITE	"	-
57	1	RTCEXIST	non-zero if Real Time Clock EXISTs	-

cont...../

INTERGLBL MAP (3.01) - INTERPRETER WORK AREA (cont.)

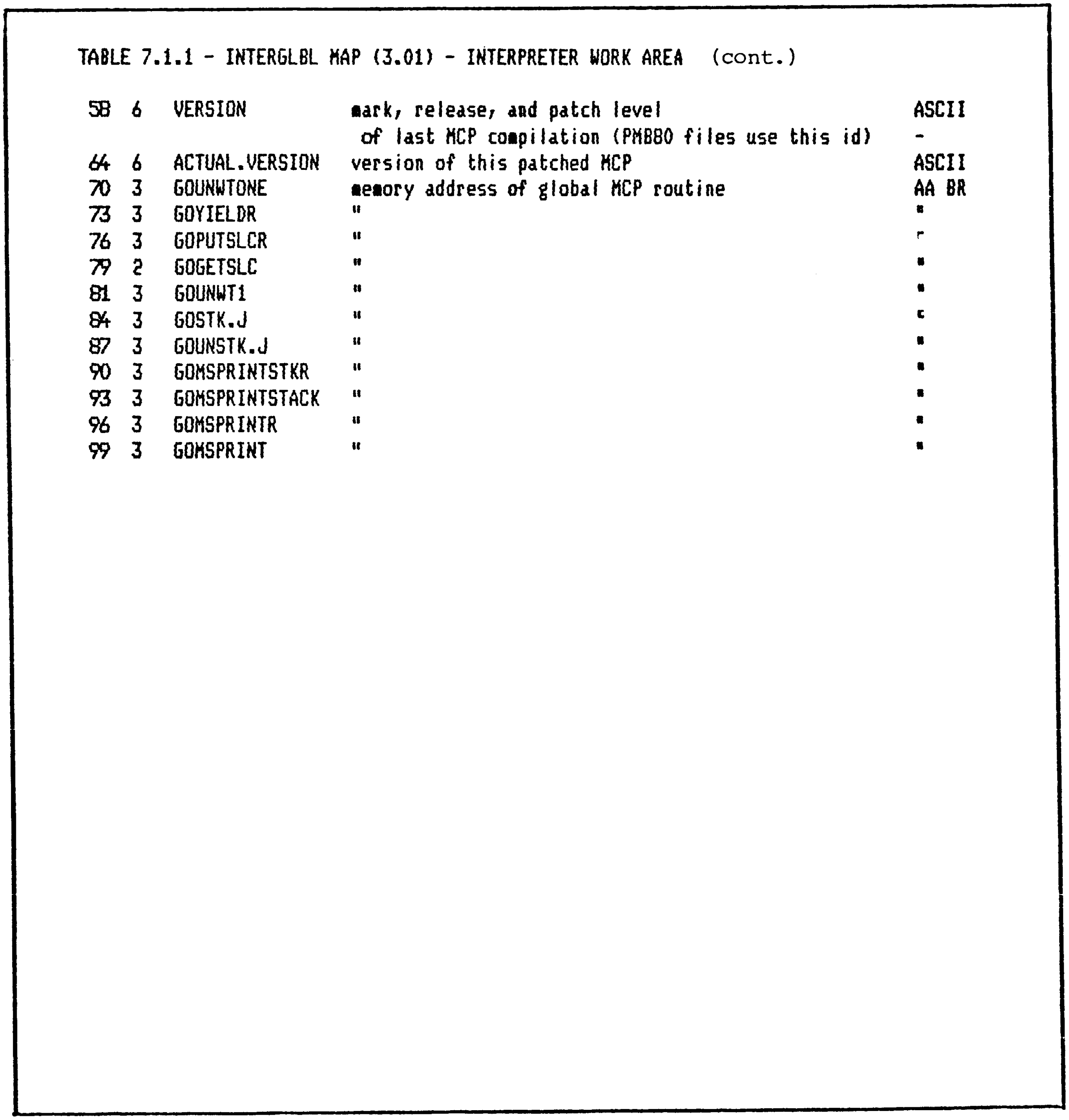


TABLE 7.1.1 - INTERGLBL MAP (3.01) - INTERPRETER WORK AREA (cont.)

58	6	VERSION	mark, release, and patch level	ASCII
			of last MCP compilation (PMB80 files use this id)	-
64	6	ACTUAL.VERSION	version of this patched MCP	ASCII
70	3	GOUNWTONE	memory address of global MCP routine	AA BR
73	3	GOYIELDR	"	"
76	3	GOPUTSLCR	"	"
79	2	GOGETSLC	"	"
81	3	GOUNWT1	"	"
84	3	GOSTK.J	"	"
87	3	GOUNSTK.J	"	"
90	3	GOMSPRINTSTKR	"	"
93	3	GOMSPRINTSTACK	"	"
96	3	GOMSPRINTR	"	"
99	3	GOMSPRINT	"	"

INTERGLBL MAP (3.01) - INTERPRETER WORK AREA

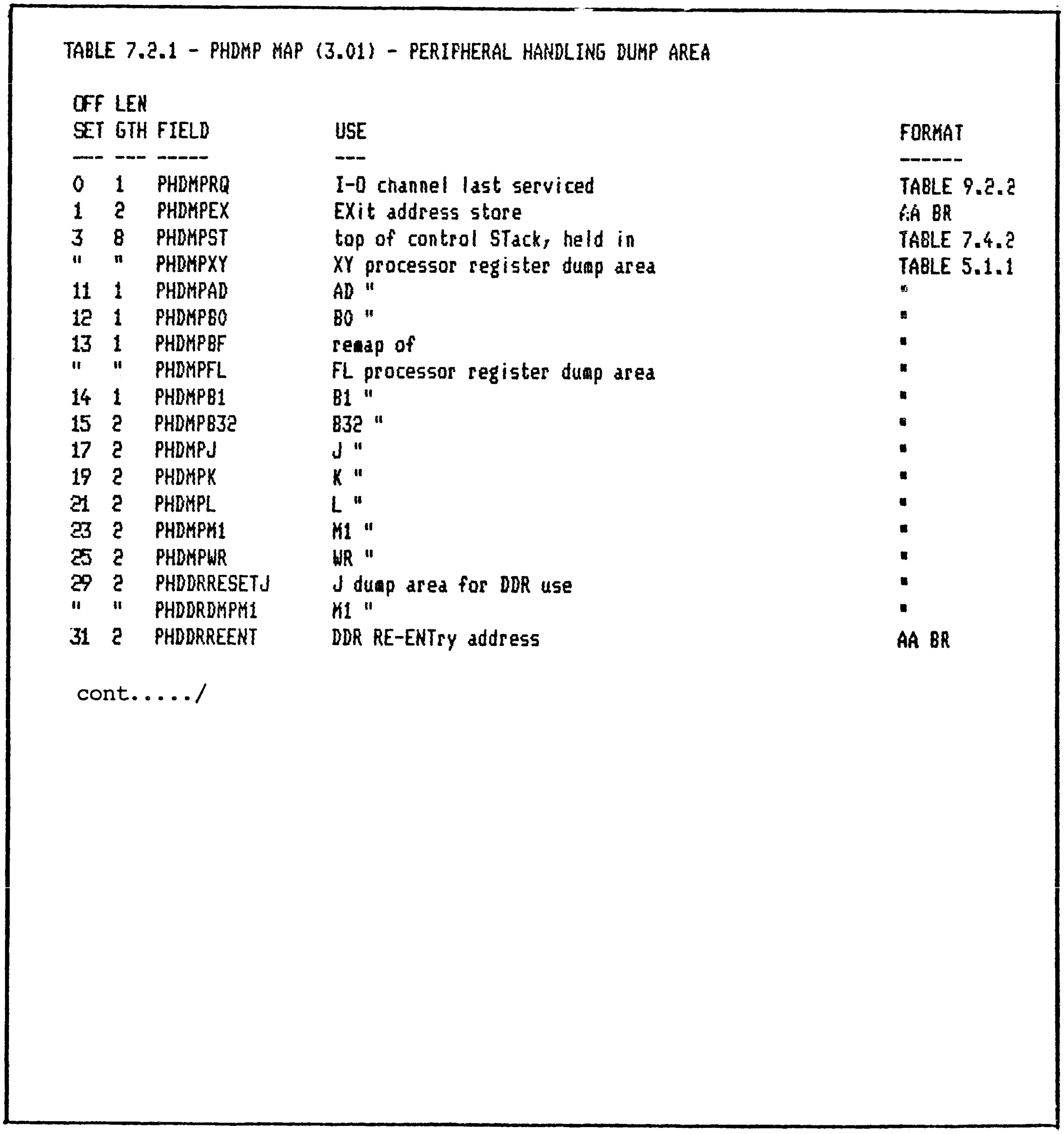


TABLE 7.2.1 - PHDMP MAP (3.01) - PERIPHERAL HANDLING DUMP AREA

OFF SET	LEN GTH	FIELD	USE	FORMAT
0	1	PHDMPRQ	I-O channel last serviced	TABLE 9.2.2
1	2	PHDMPEX	EXit address store	AA BR
3	8	PHDMPST	top of control STack, held in	TABLE 7.4.2
"	"	PHDMPXY	XY processor register dump area	TABLE 5.1.1
11	1	PHDMPAD	AD "	"
12	1	PHDMPBO	BO "	"
13	1	PHDMPBF	remap of	"
"	"	PHDMPFL	FL processor register dump area	"
14	1	PHDMPB1	B1 "	"
15	2	PHDMPB32	B32 "	"
17	2	PHDMPJ	J "	"
19	2	PHDMPK	K "	"
21	2	PHDMPL	L "	"
23	2	PHDMPM1	M1 "	"
25	2	PHDMPWR	WR "	"
29	2	PHDDRRESETJ	J dump area for DDR use	"
"	"	PHDDRDMPM1	M1 "	"
31	2	PHDDRREENT	DDR RE-ENTry address	AA BR

cont...../

PHDMP MAP (3.01) - PERIPHERAL HANDLING DUMP AREA

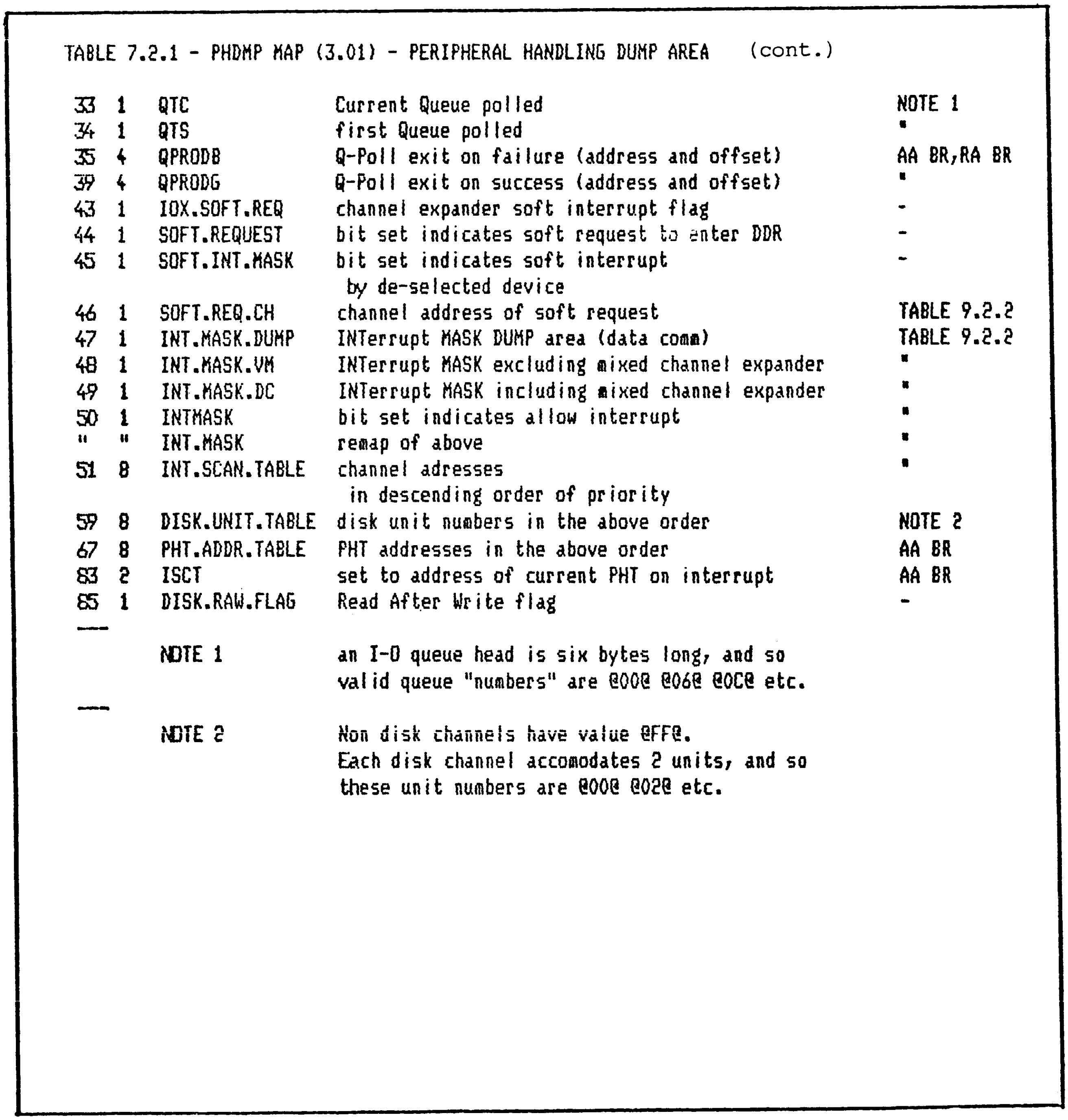


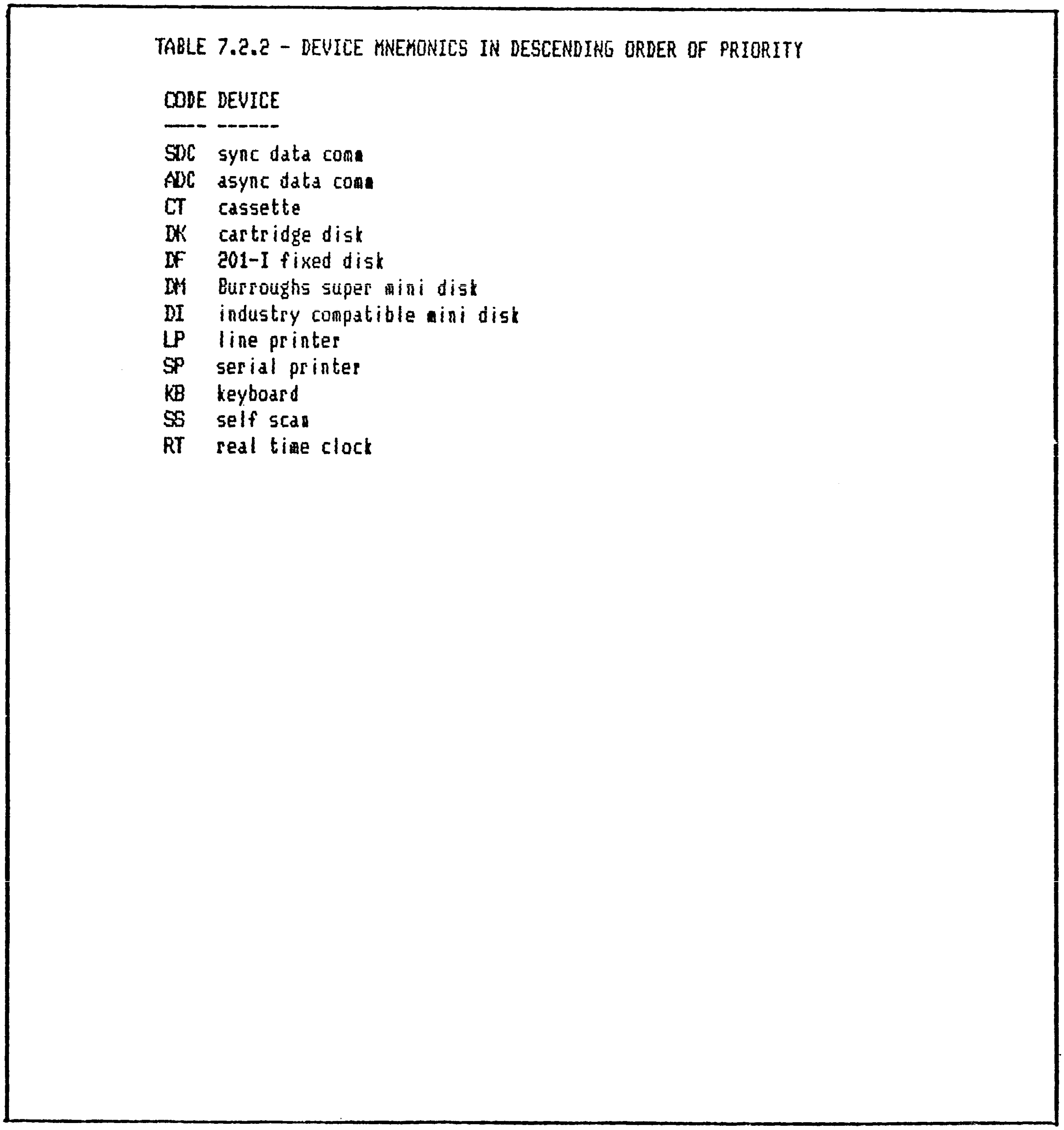
TABLE 7.2.1 - PHDMP MAP (3.01) - PERIPHERAL HANDLING DUMP AREA (cont.)

33	1	QTC	Current Queue polled	NOTE 1
34	1	QTS	first Queue polled	•
35	4	QPRODB	Q-Poll exit on failure (address and offset)	AA BR,RA BR
39	4	QPRODG	Q-Poll exit on success (address and offset)	•
43	1	IOX.SOFT.REQ	channel expander soft interrupt flag	-
44	1	SOFT.REQUEST	bit set indicates soft request to enter DDR	-
45	1	SOFT.INT.MASK	bit set indicates soft interrupt by de-selected device	-
46	1	SOFT.REQ.CH	channel address of soft request	TABLE 9.2.2
47	1	INT.MASK.DUMP	INTerrupt MASK DUMP area (data comm)	TABLE 9.2.2
48	1	INT.MASK.VM	INTerrupt MASK excluding mixed channel expander	•
49	1	INT.MASK.DC	INTerrupt MASK including mixed channel expander	•
50	1	INTMASK	bit set indicates allow interrupt	•
"	"	INT.MASK	remap of above	•
51	8	INT.SCAN.TABLE	channel adresses in descending order of priority	•
59	8	DISK.UNIT.TABLE	disk unit numbers in the above order	NOTE 2
67	8	PHT.ADDR.TABLE	PHT addresses in the above order	AA BR
83	2	ISCT	set to address of current PHT on interrupt	AA BR
85	1	DISK.RAW.FLAG	Read After Write flag	-

NOTE 1 an I-O queue head is six bytes long, and so valid queue "numbers" are @00@ @06@ @0C@ etc.

NOTE 2 Non disk channels have value @FF@.
Each disk channel accomodates 2 units, and so these unit numbers are @00@ @02@ etc.

PHDMP MAP (.301) - PERIPHERAL HANDLING DUMP AREA (cont.)



```
TABLE 7.2.2 - DEVICE MNEMONICS IN DESCENDING ORDER OF PRIORITY

CODE DEVICE
---- ------

SDC  sync data comm
ADC  async data comm
CT   cassette
DK   cartridge disk
DF   201-I fixed disk
DM   Burroughs super mini disk
DI   industry compatible mini disk
LP   line printer
SP   serial printer
KB   keyboard
SS   self scan
RT   real time clock
```

DEVICE MNEMONICS IN DESCENDING ORDER OF PRIORITY

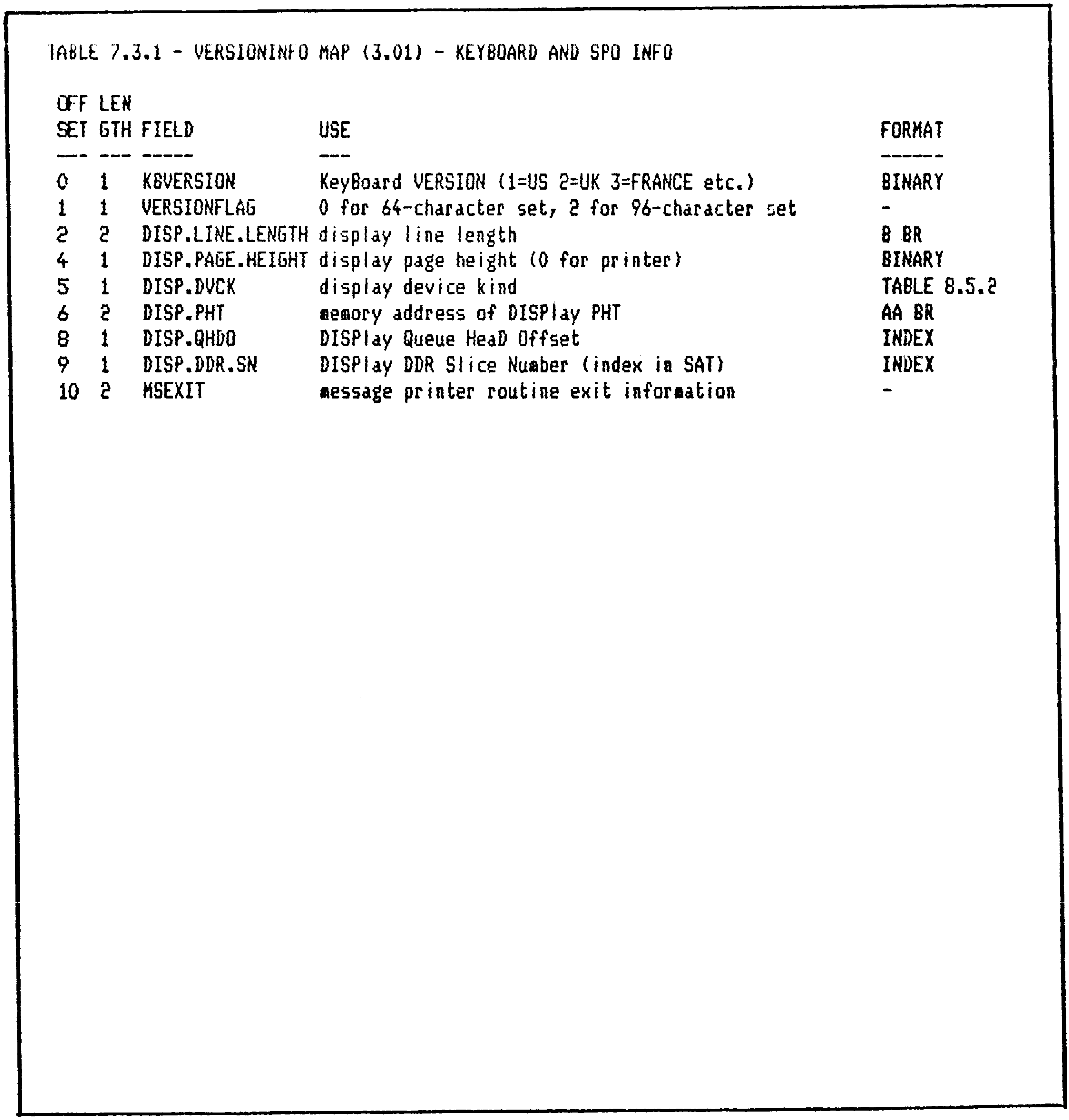


TABLE 7.3.1 - VERSIONINFO MAP (3.01) - KEYBOARD AND SPO INFO

OFF	LEN			
SET	GTH	FIELD	USE	FORMAT
0	1	KBVERSION	KeyBoard VERSION (1=US 2=UK 3=FRANCE etc.)	BINARY
1	1	VERSIONFLAG	0 for 64-character set, 2 for 96-character set	-
2	2	DISP.LINE.LENGTH	display line length	B BR
4	1	DISP.PAGE.HEIGHT	display page height (0 for printer)	BINARY
5	1	DISP.DVCK	display device kind	TABLE 8.5.2
6	2	DISP.PHT	memory address of DISPlay PHT	AA BR
8	1	DISP.QHDO	DISPlay Queue HeaD Offset	INDEX
9	1	DISP.DDR.SN	DISPlay DDR Slice Number (index in SAT)	INDEX
10	2	MSEXIT	message printer routine exit information	-

VERSIONINFO MAP (3.01) - KEYBOARD AND SPO INFO

TABLE 7.4.1 - DIAGCBUF MAP (3.01) - GLOBAL DIAGNOSTIC INFO

OFF	LEN			
SET	GTH	FIELD	USE	FORMAT
0	1	DIAGINDEX	index of next one byte entry in DIAGCIRC	INDEX
1	32	DIAGCIRC	circular buffer of one-byte DIAGNOSTICS	TABLE 5.2.=
33	-	DIAGCEND	-	LABEL
33	8	XYSAVE	top of control stack on exit from EPAR	TABLE 7.4.2
41	2	JSAVE	pointer to base of SLICE or CODE SEGMENT	AA BR
43	2	LSAVE	pointer to base of Task Control Block (TCB)	AA BR
435	1	LIGHTSW	@FF@ indicates that the console lights	-
			are available for diagnostic information	-
			i.e. the keyboard is not enabled	-

TABLE 7.4.2 - TOP FOUR BYTES OF CONTROL STACK (RESTART ADDRESS)

ENTRY CODE	SLICE or SEGMENT	OFFSET	RESULTING BRANCH
ONE BYTE	ONE BYTE	TWO BYTES REVERSED	
>=@80@	SEGMENT NUMBER	SEGMENT OFFSET	TO INTERPRETER SEGMENT
=@3F@	(>@20@)INDEX IN SAT	SEGMENT OFFSET	TO SINGLE SEGMENT SLICE
=@3F@	(=@20@)	ABSOLUTE ADDRESS	TO GLOBAL MCP CODE
<=@3E@)	(>@20@)INDEX IN SAT	SEGMENT OFFSET	TO MCP SLICE SEGMENT
SEGMENT)	(@1E@-@02@) TCB INDEX	SEGMENT OFFSET	TO TASK SEGMENT
NUMBER)	(=@00@) BAILIFF	SEGMENT OFFSET	TO BAILIFF SEGMENT

DIAGCBUF MAP (3.01) - GLOBAL DIAGNOSTIC INFO and
TOP FOUR BYTES OF CONTROL STACK (RESTART ADDRESS)

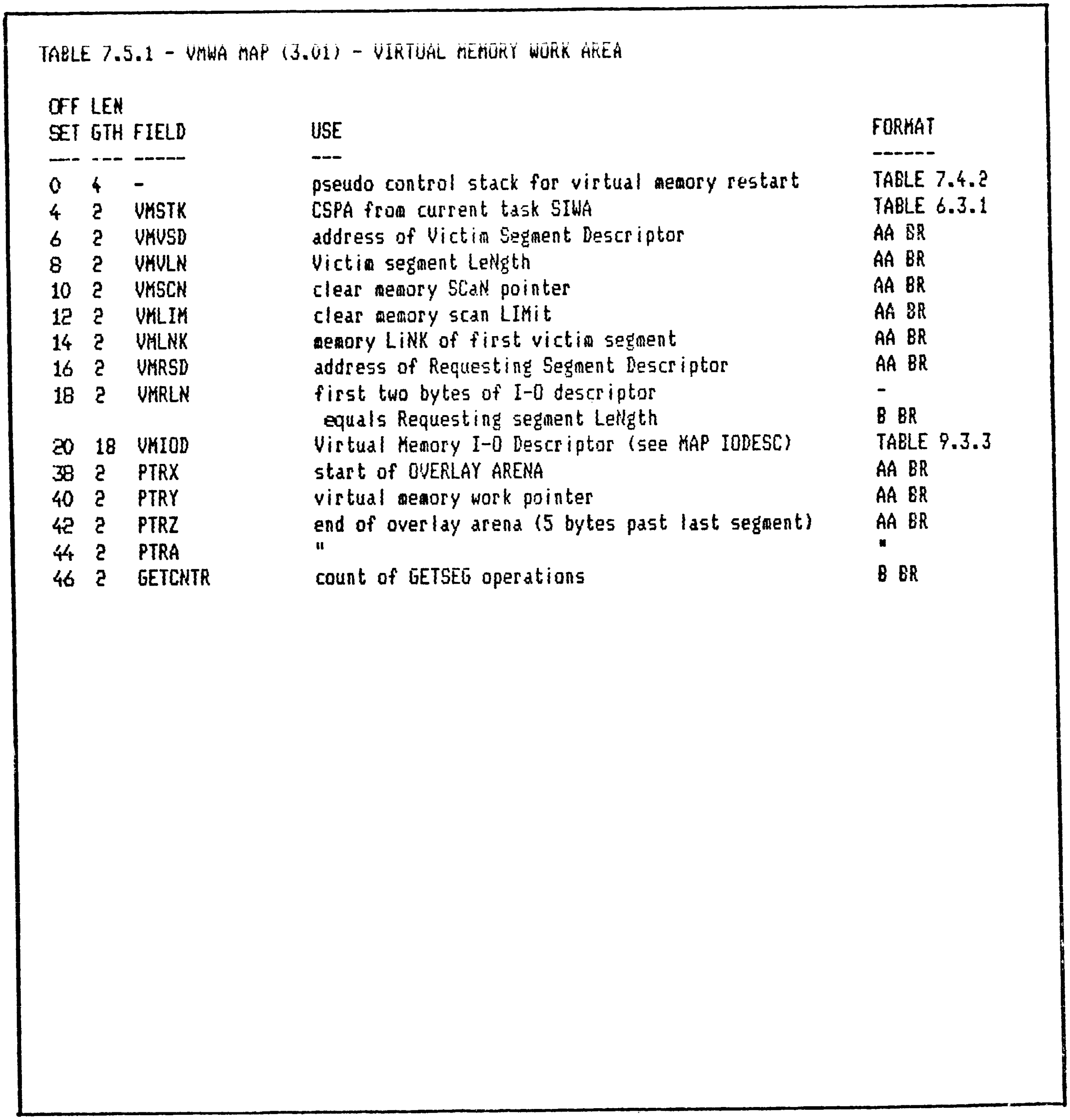


TABLE 7.5.1 - VMWA MAP (3.01) - VIRTUAL MEMORY WORK AREA

OFF SET	LEN GTH	FIELD	USE	FORMAT
0	4	-	pseudo control stack for virtual memory restart	TABLE 7.4.2
4	2	VMSTK	CSPA from current task SIWA	TABLE 6.3.1
6	2	VMVSD	address of Victim Segment Descriptor	AA BR
8	2	VMVLN	Victim segment LeNgth	AA BR
10	2	VMSCN	clear memory SCaN pointer	AA BR
12	2	VMLIM	clear memory scan LIMit	AA BR
14	2	VMLNK	memory LiNK of first victim segment	AA BR
16	2	VMRSD	address of Requesting Segment Descriptor	AA BR
18	2	VMRLN	first two bytes of I-O descriptor	-
			equals Requesting segment LeNgth	B BR
20	18	VMIOD	Virtual Memory I-O Descriptor (see MAP IODESC)	TABLE 9.3.3
38	2	PTRX	start of OVERLAY ARENA	AA BR
40	2	PTRY	virtual memory work pointer	AA BR
42	2	PTRZ	end of overlay arena (5 bytes past last segment)	AA BR
44	2	PTRA	"	"
46	2	GETCNTR	count of GETSEG operations	B BR

VMWA MAP (3.01) - VIRTUAL MEMORY WORK AREA

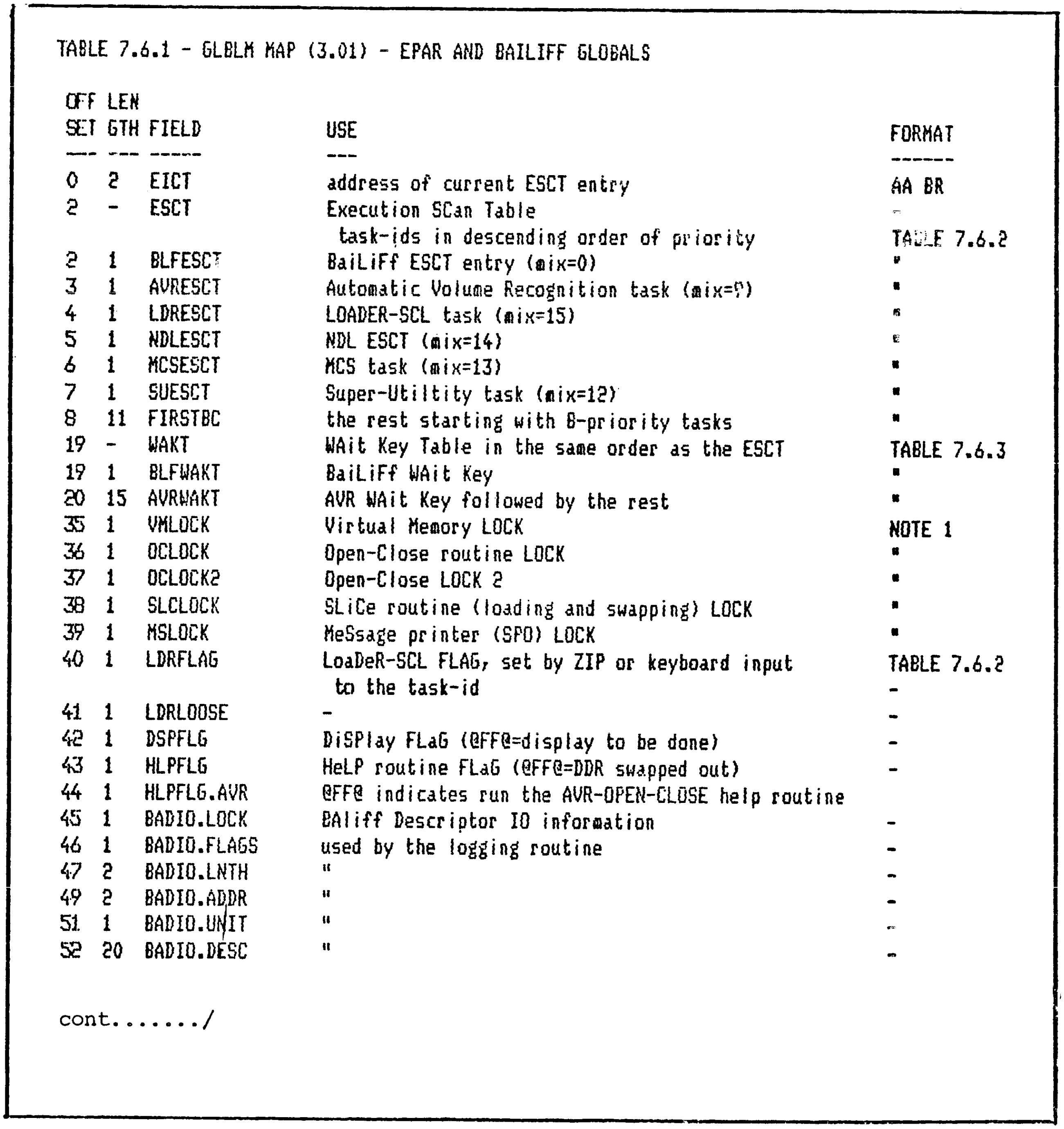


TABLE 7.6.1 - GLBLM MAP (3.01) - EPAR AND BAILIFF GLOBALS

OFF SET	LEN GTH	FIELD	USE	FORMAT
0	2	EICT	address of current ESCT entry	AA BR
2	-	ESCT	Execution SCan Table task-ids in descending order of priority	- TABLE 7.6.2
2	1	BLFESCT	BaiLiFf ESCT entry (mix=0)	"
3	1	AVRESCT	Automatic Volume Recognition task (mix=?)	"
4	1	LDRESCT	LOADER-SCL task (mix=15)	"
5	1	NDLESCT	NDL ESCT (mix=14)	"
6	1	MCSESCT	MCS task (mix=13)	"
7	1	SUESCT	Super-Utiltity task (mix=12)	"
8	11	FIRSTBC	the rest starting with B-priority tasks	"
19	-	WAKT	WAit Key Table in the same order as the ESCT	TABLE 7.6.3
19	1	BLFWAKT	BaiLiFf WAit Key	"
20	15	AVRWAKT	AVR WAit Key followed by the rest	"
35	1	VMLOCK	Virtual Memory LOCK	NOTE 1
36	1	OCLOCK	Open-Close routine LOCK	"
37	1	OCLOCK2	Open-Close LOCK 2	"
38	1	SLCLOCK	SLiCe routine (loading and swapping) LOCK	"
39	1	MSLOCK	MeSsage printer (SPO) LOCK	"
40	1	LDRFLAG	LoaDeR-SCL FLAG, set by ZIP or keyboard input to the task-id	TABLE 7.6.2 -
41	1	LDRLOOSE	-	-
42	1	DSPFLG	DiSPlay FLaG (@FF@=display to be done)	-
43	1	HLPFLG	HeLP routine FLaG (@FF@=DDR swapped out)	-
44	1	HLPFLG.AVR	@FF@ indicates run the AVR-OPEN-CLOSE help routine	
45	1	BADIO.LOCK	BAliff Descriptor IO information	-
46	1	BADIO.FLAGS	used by the logging routine	-
47	2	BADIO.LNTH	"	-
49	2	BADIO.ADDR	"	-
51	1	BADIO.UNIT	"	-
52	20	BADIO.DESC	"	-

cont......./

GLBLM MAP (3.01) - EPAR AND BAILIFF GLOBALS

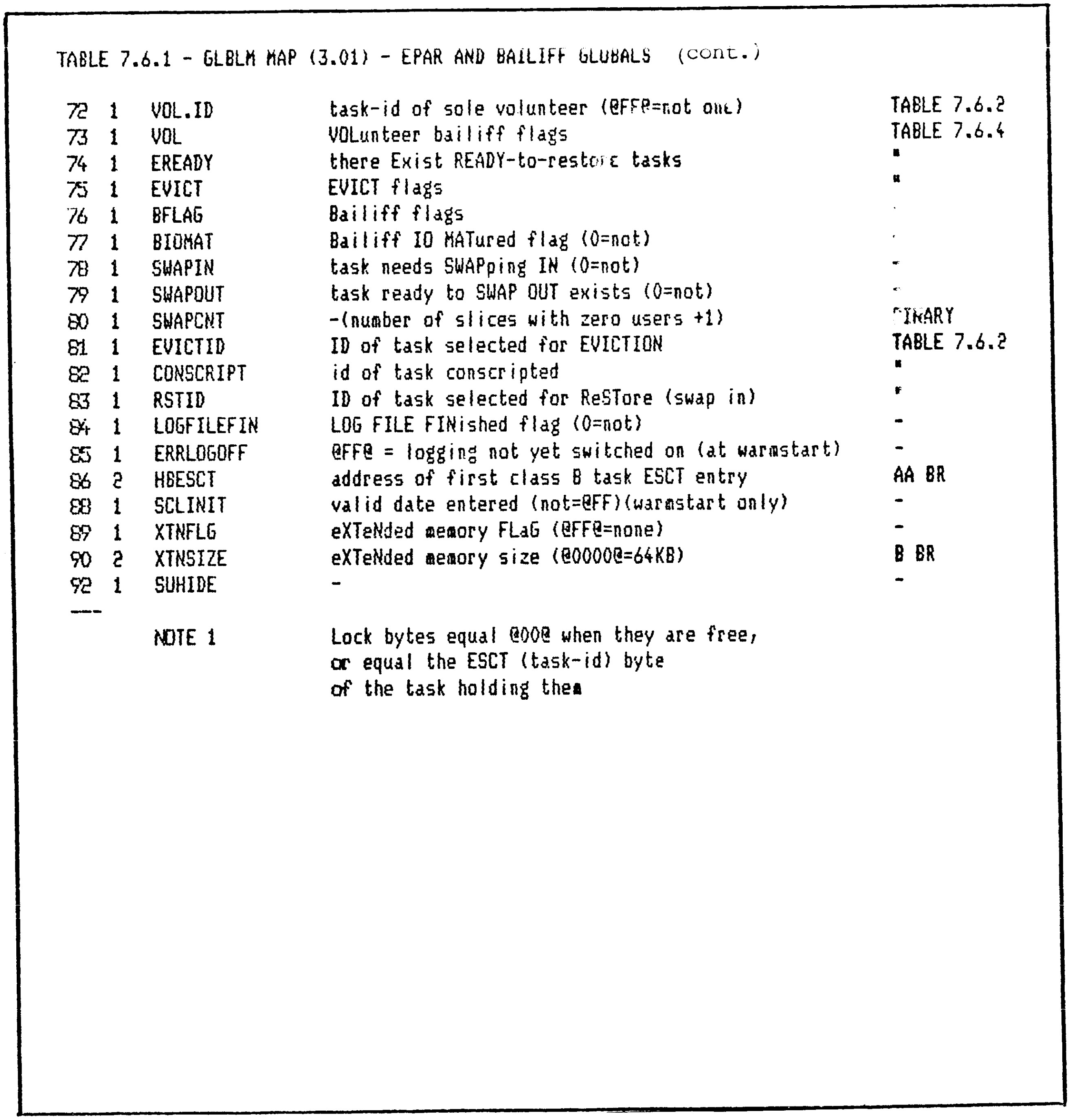


TABLE 7.6.1 - GLBLM MAP (3.01) - EPAR AND BAILIFF GLOBALS (cont.)

72	1	VOL.ID	task-id of sole volunteer (@FF@=not one)	TABLE 7.6.2
73	1	VOL	VOLunteer bailiff flags	TABLE 7.6.4
74	1	EREADY	there Exist READY-to-restore tasks	"
75	1	EVICT	EVICT flags	"
76	1	BFLAG	Bailiff flags	.
77	1	BIOMAT	Bailiff IO MATured flag (0=not)	.
78	1	SWAPIN	task needs SWAPping IN (0=not)	-
79	1	SWAPOUT	task ready to SWAP OUT exists (0=not)	-
80	1	SWAPCNT	-(number of slices with zero users +1)	BINARY
81	1	EVICTID	ID of task selected for EVICTION	TABLE 7.6.2
82	1	CONSCRIPT	id of task conscripted	"
83	1	RSTID	ID of task selected for ReSTore (swap in)	"
84	1	LOGFILEFIN	LOG FILE FINished flag (0=not)	-
85	1	ERRLOGOFF	@FF@ = logging not yet switched on (at warmstart)	-
86	2	HBESCT	address of first class B task ESCT entry	AA BR
88	1	SCLINIT	valid date entered (not=@FF)(warmstart only)	-
89	1	XTNFLG	eXTeNded memory FLaG (@FF@=none)	-
90	2	XTNSIZE	eXTeNded memory size (@0000@=64KB)	B BR
92	1	SUHIDE	-	-

NOTE 1 Lock bytes equal @00@ when they are free,
or equal the ESCT (task-id) byte
of the task holding them

GLBLM MAP (3.01) - EPAR AND BAILIFF GLOBALS (cont.)

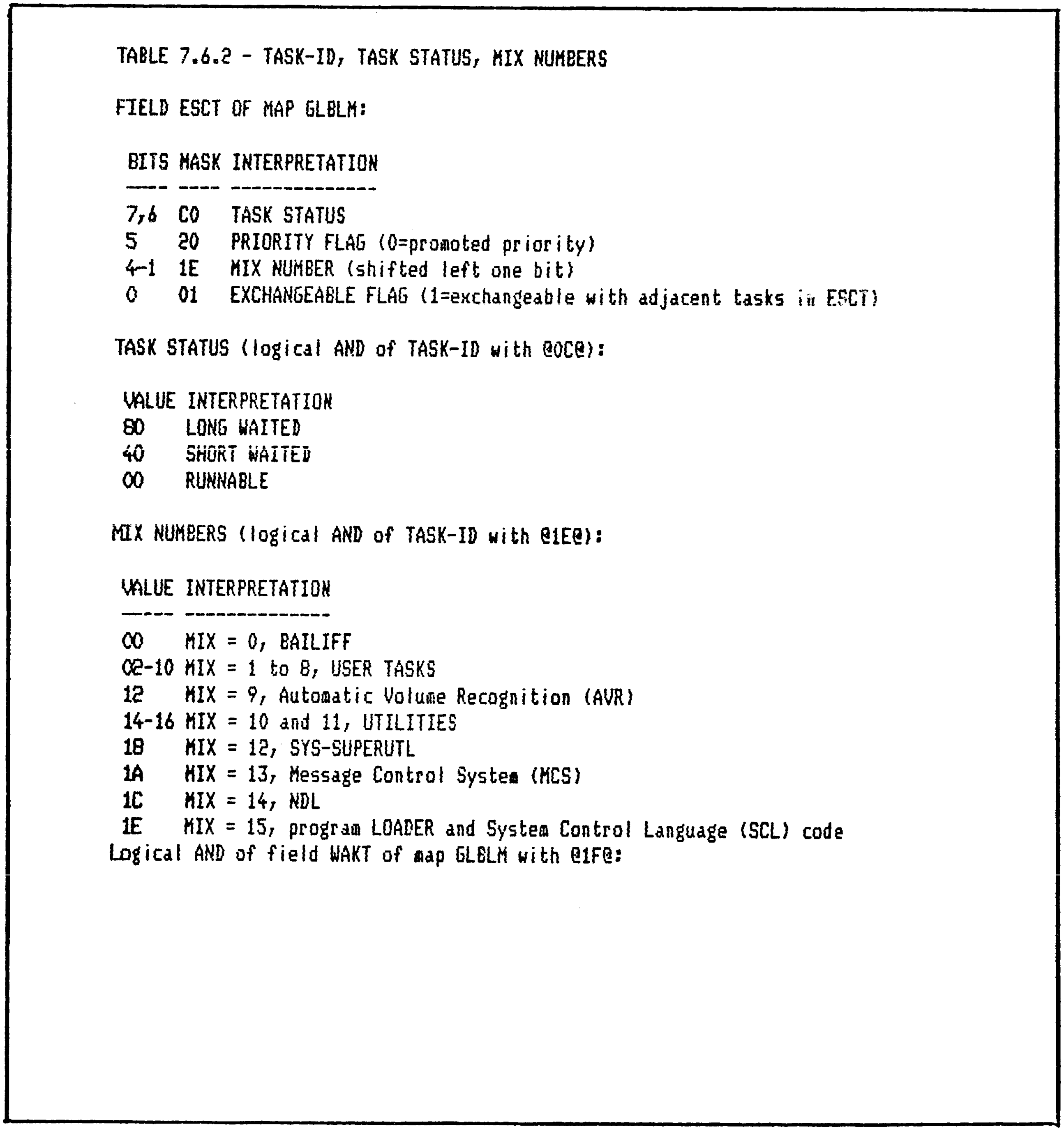


TABLE 7.6.2 - TASK-ID, TASK STATUS, MIX NUMBERS

FIELD ESCT OF MAP GLBLM:

BITS	MASK	INTERPRETATION
7,6	C0	TASK STATUS
5	20	PRIORITY FLAG (0=promoted priority)
4-1	1E	MIX NUMBER (shifted left one bit)
0	01	EXCHANGEABLE FLAG (1=exchangeable with adjacent tasks in ESCT)

TASK STATUS (logical AND of TASK-ID with @0C@):

VALUE	INTERPRETATION
80	LONG WAITED
40	SHORT WAITED
00	RUNNABLE

MIX NUMBERS (logical AND of TASK-ID with @1E@):

VALUE	INTERPRETATION
00	MIX = 0, BAILIFF
02-10	MIX = 1 to 8, USER TASKS
12	MIX = 9, Automatic Volume Recognition (AVR)
14-16	MIX = 10 and 11, UTILITIES
18	MIX = 12, SYS-SUPERUTL
1A	MIX = 13, Message Control System (MCS)
1C	MIX = 14, NDL
1E	MIX = 15, program LOADER and System Control Language (SCL) code

Logical AND of field WAKT of map GLBLM with @1F@:

TASK-ID, TASK STATUS, MIX NUMBERS

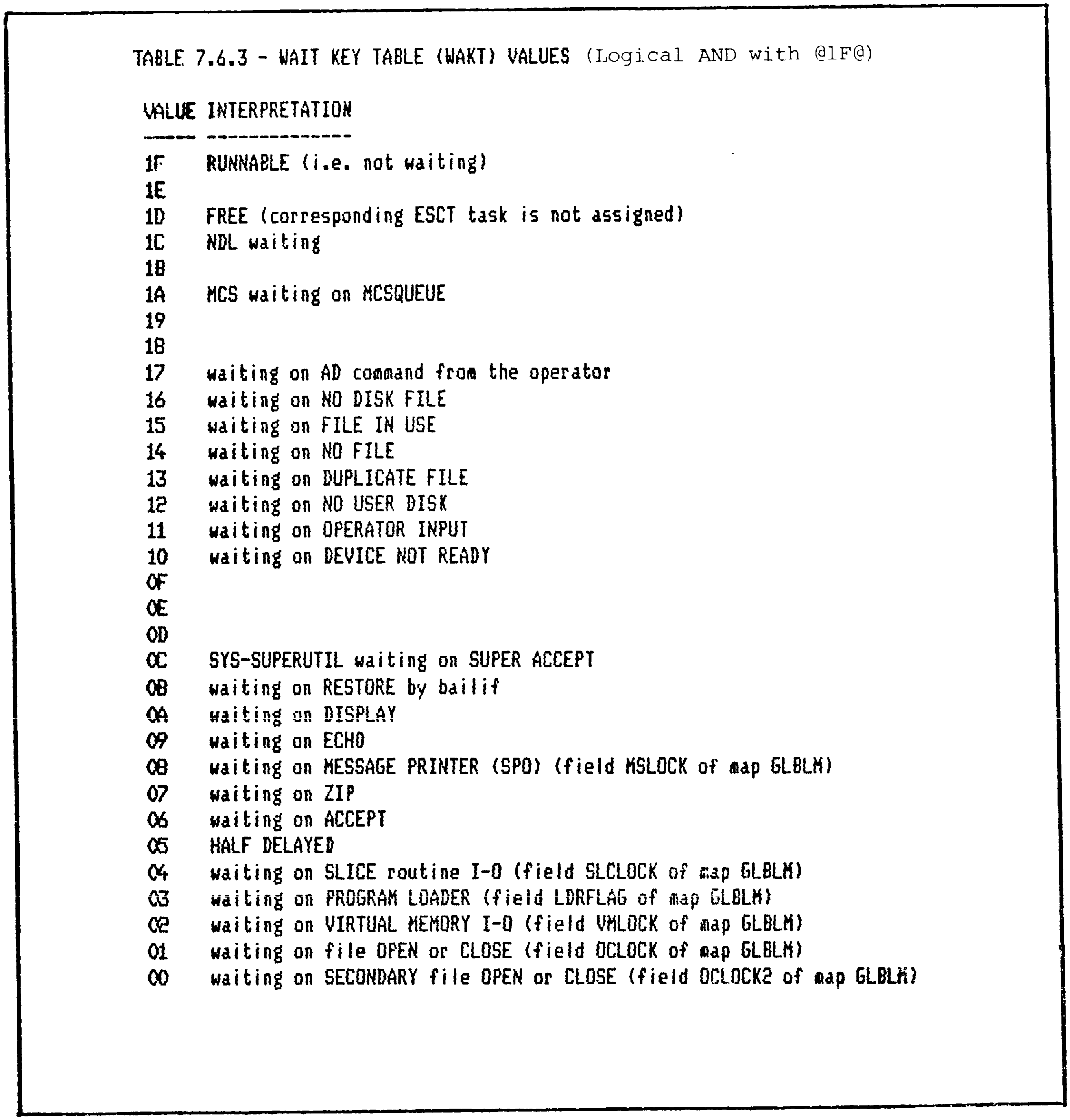
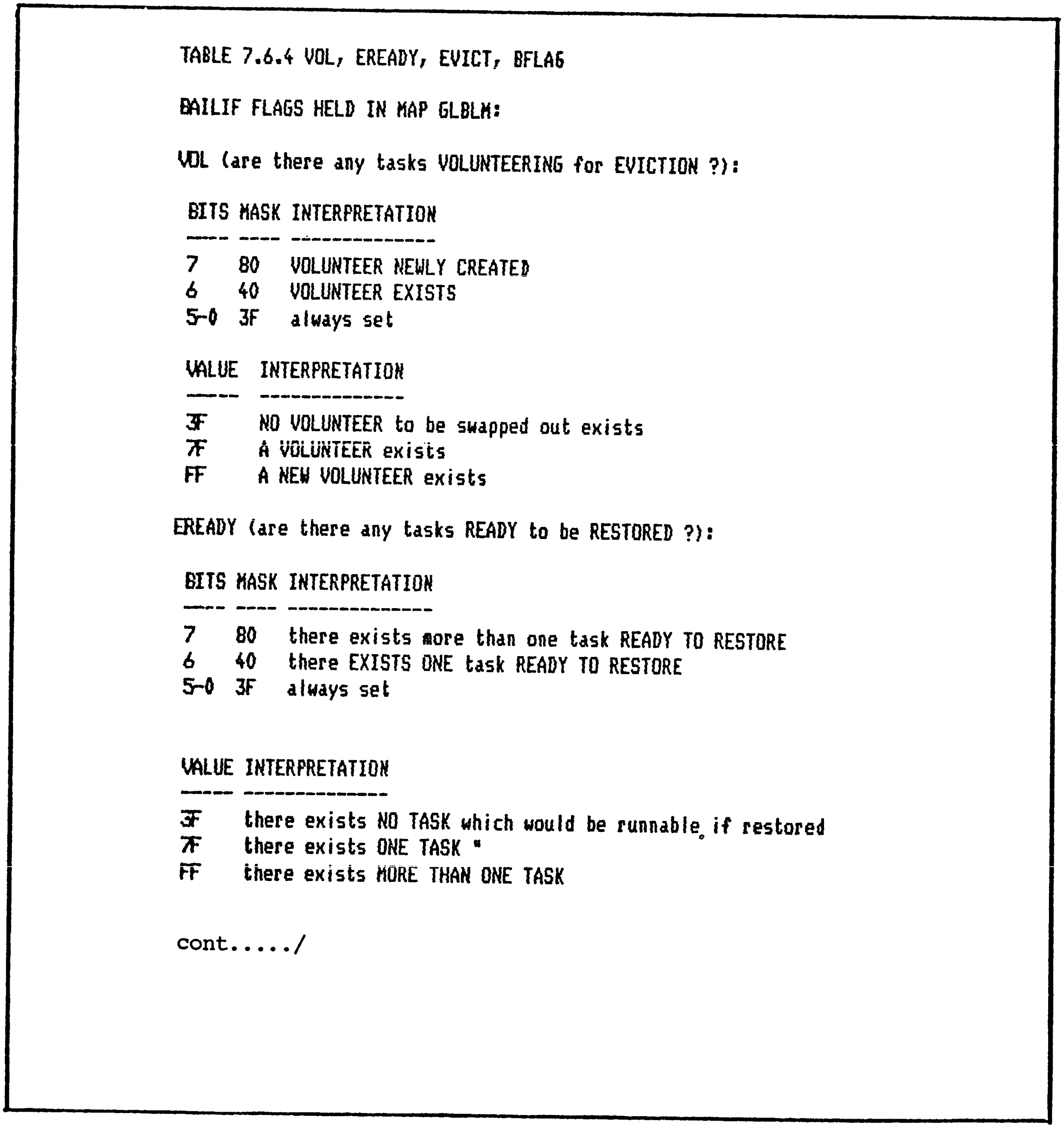


TABLE 7.6.3 - WAIT KEY TABLE (WAKT) VALUES (Logical AND with @1F@)

VALUE	INTERPRETATION
1F	RUNNABLE (i.e. not waiting)
1E	
1D	FREE (corresponding ESCT task is not assigned)
1C	NDL waiting
1B	
1A	MCS waiting on MCSQUEUE
19	
18	
17	waiting on AD command from the operator
16	waiting on NO DISK FILE
15	waiting on FILE IN USE
14	waiting on NO FILE
13	waiting on DUPLICATE FILE
12	waiting on NO USER DISK
11	waiting on OPERATOR INPUT
10	waiting on DEVICE NOT READY
0F	
0E	
0D	
0C	SYS-SUPERUTIL waiting on SUPER ACCEPT
0B	waiting on RESTORE by bailif
0A	waiting on DISPLAY
09	waiting on ECHO
08	waiting on MESSAGE PRINTER (SPO) (field MSLOCK of map GLBLM)
07	waiting on ZIP
06	waiting on ACCEPT
05	HALF DELAYED
04	waiting on SLICE routine I-O (field SLCLOCK of map GLBLM)
03	waiting on PROGRAM LOADER (field LDRFLAG of map GLBLM)
02	waiting on VIRTUAL MEMORY I-O (field VMLOCK of map GLBLM)
01	waiting on file OPEN or CLOSE (field OCLOCK of map GLBLM)
00	waiting on SECONDARY file OPEN or CLOSE (field OCLOCK2 of map GLBLM)

WAIT KEY TABLE (WAKT) VALUES



```
TABLE 7.6.4 VOL, EREADY, EVICT, BFLAG

BAILIF FLAGS HELD IN MAP GLBLM:

VOL (are there any tasks VOLUNTEERING for EVICTION ?):

 BITS MASK INTERPRETATION
 ---- ---- --------------
 7    80   VOLUNTEER NEWLY CREATED
 6    40   VOLUNTEER EXISTS
 5-0  3F   always set

 VALUE  INTERPRETATION
 -----  --------------
 3F     NO VOLUNTEER to be swapped out exists
 7F     A VOLUNTEER exists
 FF     A NEW VOLUNTEER exists

EREADY (are there any tasks READY to be RESTORED ?):

 BITS MASK INTERPRETATION
 ---- ---- --------------
 7    80   there exists more than one task READY TO RESTORE
 6    40   there EXISTS ONE task READY TO RESTORE
 5-0  3F   always set


 VALUE INTERPRETATION
 ----- --------------
 3F    there exists NO TASK which would be runnable if restored
 7F    there exists ONE TASK "
 FF    there exists MORE THAN ONE TASK

cont...../
```

VOL, EREADY, EVICT, BFLAG

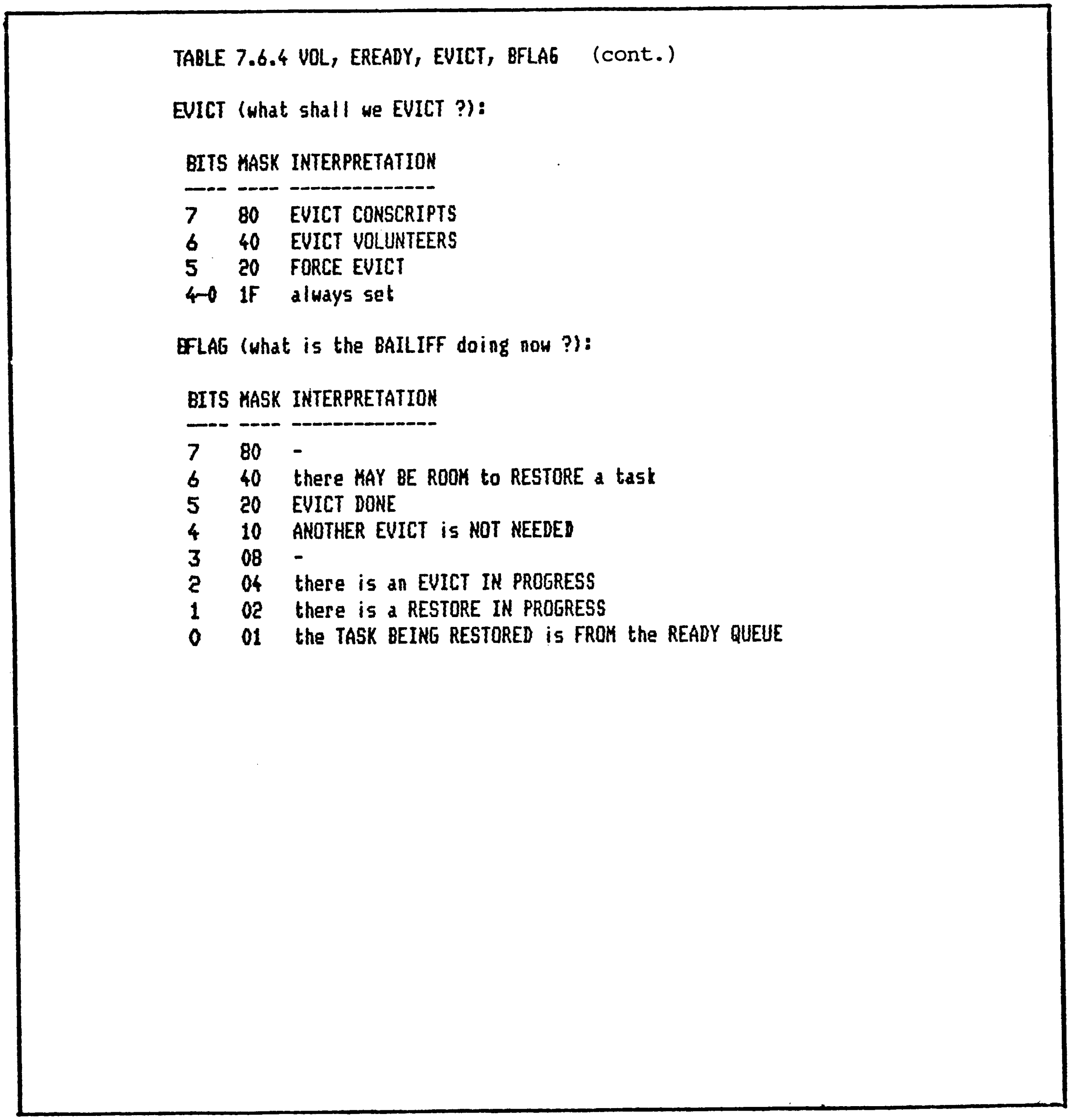


TABLE 7.6.4 VOL, EREADY, EVICT, BFLAG (cont.)

EVICT (what shall we EVICT ?):

BITS	MASK	INTERPRETATION
7	80	EVICT CONSCRIPTS
6	40	EVICT VOLUNTEERS
5	20	FORCE EVICT
4-0	1F	always set

BFLAG (what is the BAILIFF doing now ?):

BITS	MASK	INTERPRETATION
7	80	-
6	40	there MAY BE ROOM to RESTORE a task
5	20	EVICT DONE
4	10	ANOTHER EVICT is NOT NEEDED
3	08	-
2	04	there is an EVICT IN PROGRESS
1	02	there is a RESTORE IN PROGRESS
0	01	the TASK BEING RESTORED is FROM the READY QUEUE

VOL, EREADY, EVICE, BFLAG (cont.)

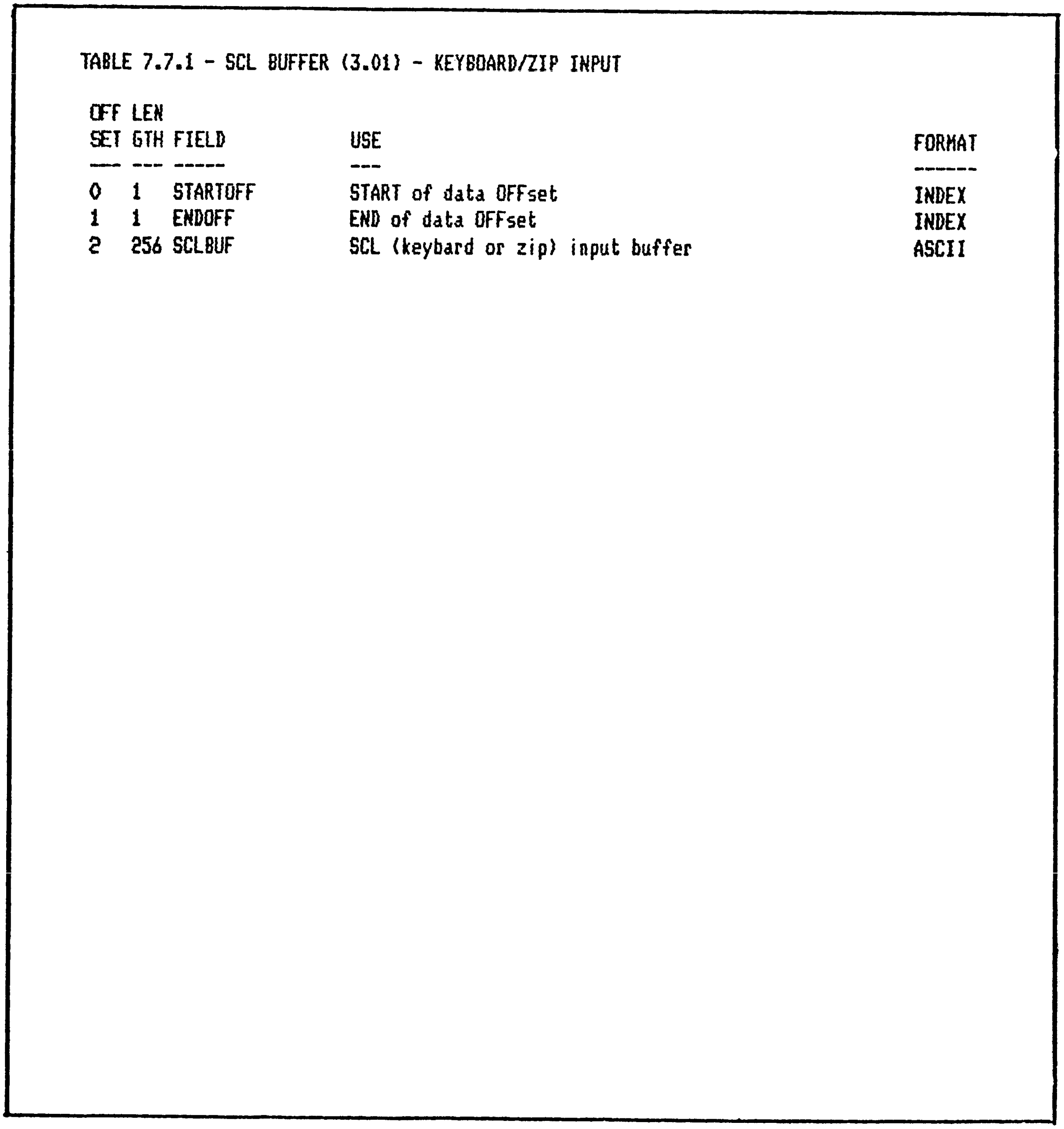


TABLE 7.7.1 - SCL BUFFER (3.01) - KEYBOARD/ZIP INPUT

OFF	LEN			
SET	GTH	FIELD	USE	FORMAT
0	1	STARTOFF	START of data OFFset	INDEX
1	1	ENDOFF	END of data OFFset	INDEX
2	256	SCLBUF	SCL (keybard or zip) input buffer	ASCII

SCL BUFFER (3.01) - KEYBOARD/ZIP INPUT

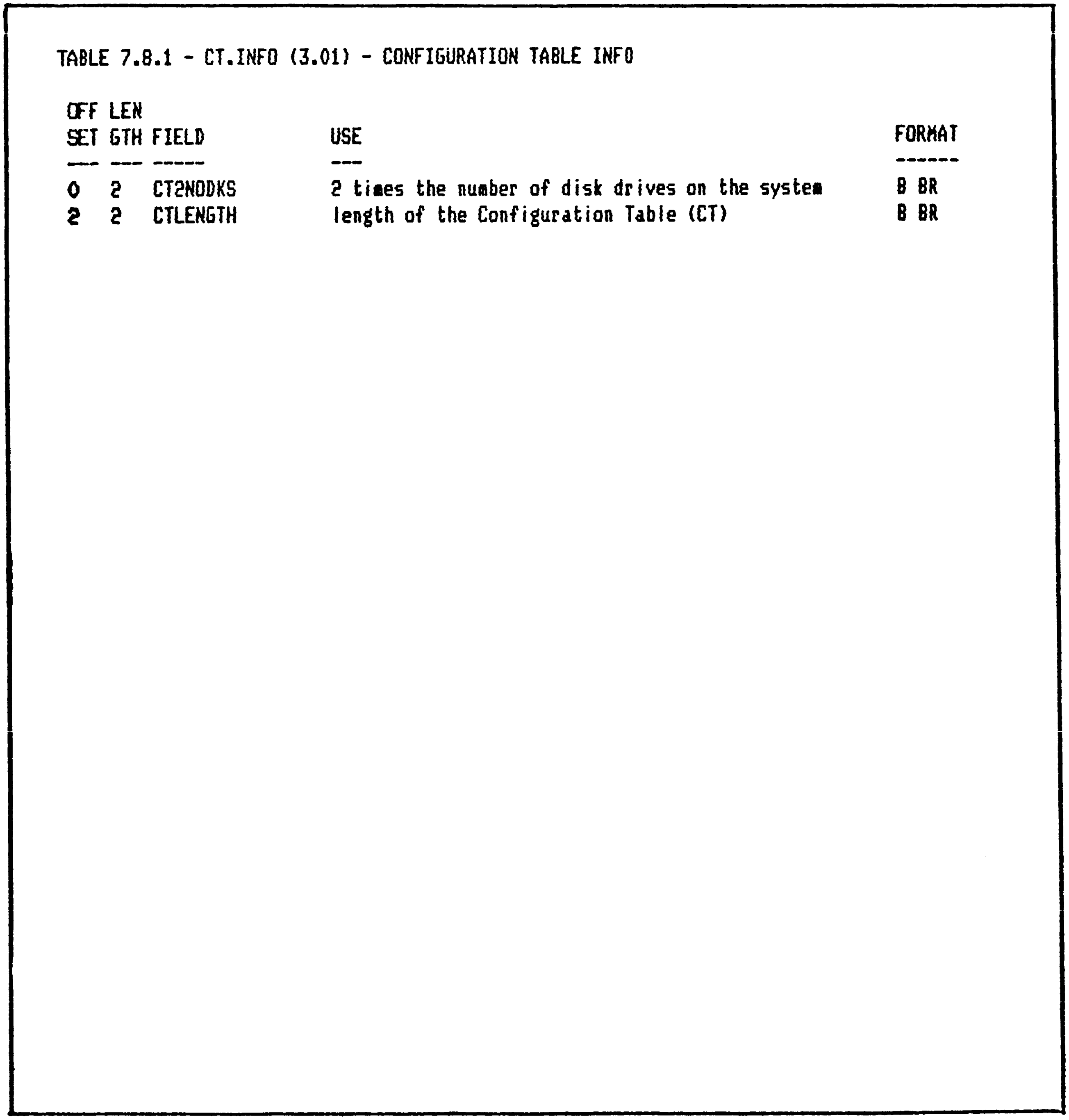


TABLE 7.8.1 - CT.INFO (3.01) - CONFIGURATION TABLE INFO

OFF	LEN			
SET	GTH	FIELD	USE	FORMAT
0	2	CT2NODKS	2 times the number of disk drives on the system	B BR
2	2	CTLENGTH	length of the Configuration Table (CT)	B BR

CT.INFO (3.01) - CONFIGURATION TABLE INFO

SECTION 8

8. TASK RUN STRUCTURES

This section explains the organisation and content of the different task structures. If an investigation of global MCP tables and memory layout reveals no cause of a failure, analysis of the contents of the task structures should be made to determine what action the tasks have most recently been performing.

As explained in section 7.6.1, the MCP maintains a set of tasks called "independent runners". Only one of these tasks will have control at a particular time. A task structure may be printed out using the PRINT TASK option of PMB80. The CURRENT parameter causes the task currently in control to be printed.

8.1 TASK ORGANISATION IN MEMORY

Tasks are accessed through the slices numbered 0-15; the Task Control Blocks. Slice numbers 0, 9 and 15 are reserved for the MCP slices BAILIFF, AVR and SCL/LOADER discussed in section 8.2. The other slices in this range accommodate "user" TCBs as required. The TCB slice number is the same as the task mix number.

The MCP tasks are single slice tasks, and the slices contain both data and microcode segments. User tasks consist of three slices: a data slice (TCB), a program S-code slice (PCB) and an interpreter microcode slice (ICB). An exception to this is SORTINTRINS which is a microcoded program and so does not need an interpreter.

A TCB is set up when a program is executed on the B80. The code and interpreter slices (PCB, TCB) for this program are created only if these slices are not already in memory.

Program s-code and interpreter microcode is re-entrant, and therefore may be used by more than one task.

8.1 TASK ORGANISATION IN MEMORY (CONT.)

The TCB slice descriptor holds the index in the Slice Address Table (SAT) of the program and interpreter slice addresses (fields SDPEO, SDIEO of Table 6.3.1 - RS MAP). These two links locate the slices in use by a task, and these slices in turn hold the Data Segment Tables (DSTs). which locate all overlayable segments associated with the task.

8.1.1 CONTENTS OF A TCB

A TCB locked slice contains a slice descriptor; an S-Interpreter Work Area (SIWA); a Control Stack; a Data Segment Table (DST), and any locked segments including File Information Blocks.

An interpreter needs a working storage area for each task which it is executing; this is the SIWA. The format of an SIWA depends on the interpreter, but always starts with an extension to the slice descriptor which provides the interface between the interpreter and the MCP, and holds the memory addresses of the items within the locked slice.

The use of the control stack is dependent on the S-machine (refer to the CMS MCP Reference Manual for details of the S-languages). The control stack is also used by the MCP to store return addresses, as explained in section 7.4.2

The DST holds the segment descriptors of all segments of data associated with this task. See Table 6.4.1 - SD MAP.

Any data segments which are not overlayable are also located within the locked TCB slice. Examples of locked segments are File Information Blocks (FIBs) for all open files (see section 8.6). The Data Stack (data segment 0) of MPLII programs is also a locked segment.

8.1.2 CONTENTS OF A PCB

A PCB locked slice contains a slice descriptor an optional preset area (CCBPA2) and a Data Segment Table.

For COBOL and RPG programs, the COP table is treated as part of the S-code (it does not change at run time). It is included in the PCB, following directly after the slice descriptor.

The DST contains the segment descriptors of all segments of s-code for the program.

8.2 SYSTEM TASKS

The system tasks are permanently present in the B80 mix although they are not evident in the response to the MX command. Detailed information about the contents of these TCBs is not usually required.

8.2.1 BAILIFF TASK

The Bailiff is task 0. The function of this task is explained in section 7.6.3. Segment 6 of slice 0 is the TASK TABLE which contains the names of the tasks is the mix along with other information (see Tables 8.2.1 and 8.2.2). This information is included in the analysis provided by the PRINT MIX option of PMB80.

8.2.2 AVR TASK

The Automatic Volume Recognition (AVR) task has mix number 9. The purpose of this task is to allow the AVR operation to run efficiently with other tasks. It is executed when peripherals (especially disks and cassettes) are brought on-line and made ready. The MCP AVR code resides in the OPENCLOSE slice (17).

8.2.3 SCL/LOADER TASK

This MCP task has mix number 15. The TCB references a large number of segments including work areas, FIBs and FPBs for code files and virtual memory files, and micro-code segments for performing various functions.

8.3 USER TASKS

With the exception of SORTINTRINS all programs create a run structure including three slices (TCB, PCB and ICB). The TCB contains the run time information of the program. It is usually necessary to understand the S-language involved to perform a detailed analysis of the TCB information. This is not, however, usually relevant to system dump analysis. The most relevant information is the exact function that the task was performing which might have caused the system failure. Consequently, only the general layout of the task structures is provided here.

8.3 USER TASKS (CONT.)

8.3.1 COBOL AND RPG TASKS

COBOL and RPG programs both use the same S-interpreter (COBOLINT), consequently their tasks have identical run structures.

Each file declared in a program has three segments in the task structure: one FPB (Table 8.5.1), one FIB (Table 8.6.1) and one segment to contain the record work area.

A file with indexed organisation has a larger FIB which includes an IFIB (Table 8.7.1) All FIBs are located within the TCB locked slice.

Working storage has segments of size set by the COBOL dollar option DSSIZE with the limitation that segments occur on 01 levels.

Program S-code is segmented as specified by the COBOL SECTION statement.

The COP table is located within the PCB locked slice between the slice descriptor and the Data Segment Table. It is called CCBPA2 on a memory dump.

8.3.2 MPLII (BIL) TASKS

MPLII programs and BIL programs (the CMS compilers) share the same interpreter (BILINTERP). In MPL, all data that is not specifically declared within an overlayable segment resides in the Data Stack (data segment 0). This segment is located in the TCB locked slice of an MPL program.

Each file declared has an FPB segment and an FIB segment in the task slice, the FIB being located within the locked TCB.

The PCB (program code slice) contains only a slice descriptor and a Data Segment Table.

8.4 COMMUNICATES AND FETCH VALUES

If the execution of a particular task causes the system to fail, then the failure is more likely to have occurred while the MCP was performing a function related to the task, than while the task's normal S-code was being interpreted.

8.4 COMMUNICATES AND FETCH VALUES (CONT.)

A task makes a request to the MCP via a "communicate" S-Instruction. The last communicate can be found in the SIWA located within the task's TCB (see fields CPA.VERB and CPA of COBOL.TCB MAP Table 8.3.1, and field CPA of MTCB MAP Table 8.3.2). The value of the communicate indicates the function that the MCP is probably performing (see Table 8.4.1). This often directs attention to a specific Device Dependent Routine or Function Dependent Routine. This may be a clue to the writing of a small test program to attempt to create a reproducible occurrence of the problem.

When the MCP completes a communicate request, it returns control to the task and puts a value in the TCB called the "fetch value" or Fetch Communicate Message (field FCM or MAP RS - Table 6.3.1). The value of this field indicates the result of the operation (see Table 8.4.2).

8.5 FILE PARAMETER BLOCK (FPB)

This data segment which is present in a program code file contains the initial values relating to a logical file. It contains sufficient information to enable a task to locate a physical file on a specific medium (see Table 8.5.1 - FPB MAP). The FPB in the code file can be changed with the MODIFY utility.

8.6 FILE INFORMATION BLOCK (FIB)

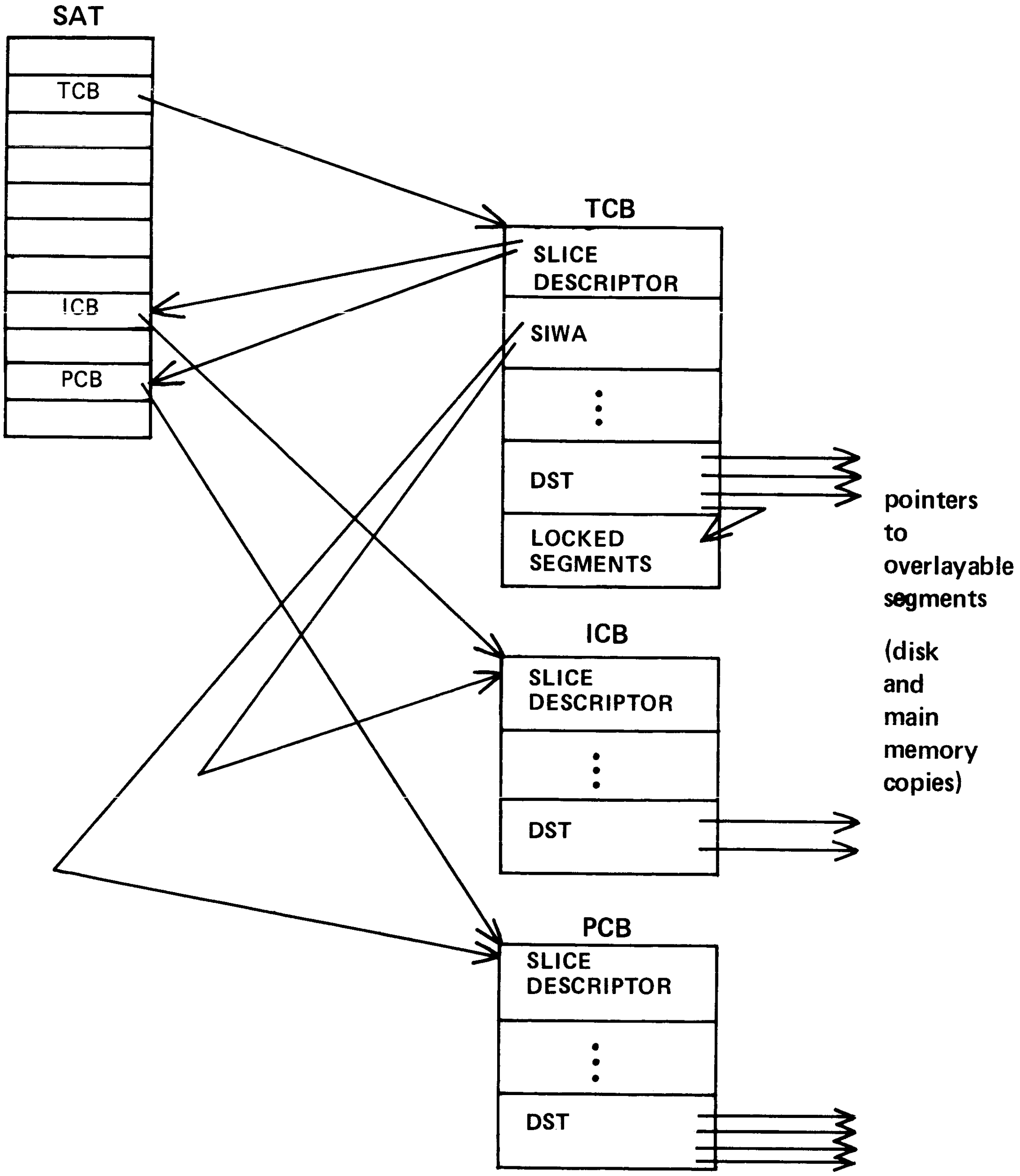
In addition to an FPB, a logical file has an FIB. If the file is closed, the information is small enough to reside in a special segment descriptor with a flags value of @48@ called a "vestigial FIB" (Table 8.6.2 VEST.FIB MAP). This descriptor resides in the DST of the TCB. When the logical file is opened, an FIB segment is created in the locked TCB slice and values inserted which reference the physical file.

An FIB includes the file buffers with an I-O descriptor for each. When an I-O operation is to be performed on the file, one of these I-O descriptors is linked to a queue of descriptors called an I-O queue. Pointers to the queue are located in the PHT of the device involved (see Section 9). The I-O descriptors should be checked for valid contents, since invalid I-O information (which may be a result of a hardware fault) is a frequent cause of system failures.

8.7 INDEXED FILE INFORMATION BLOCK (IFIB)

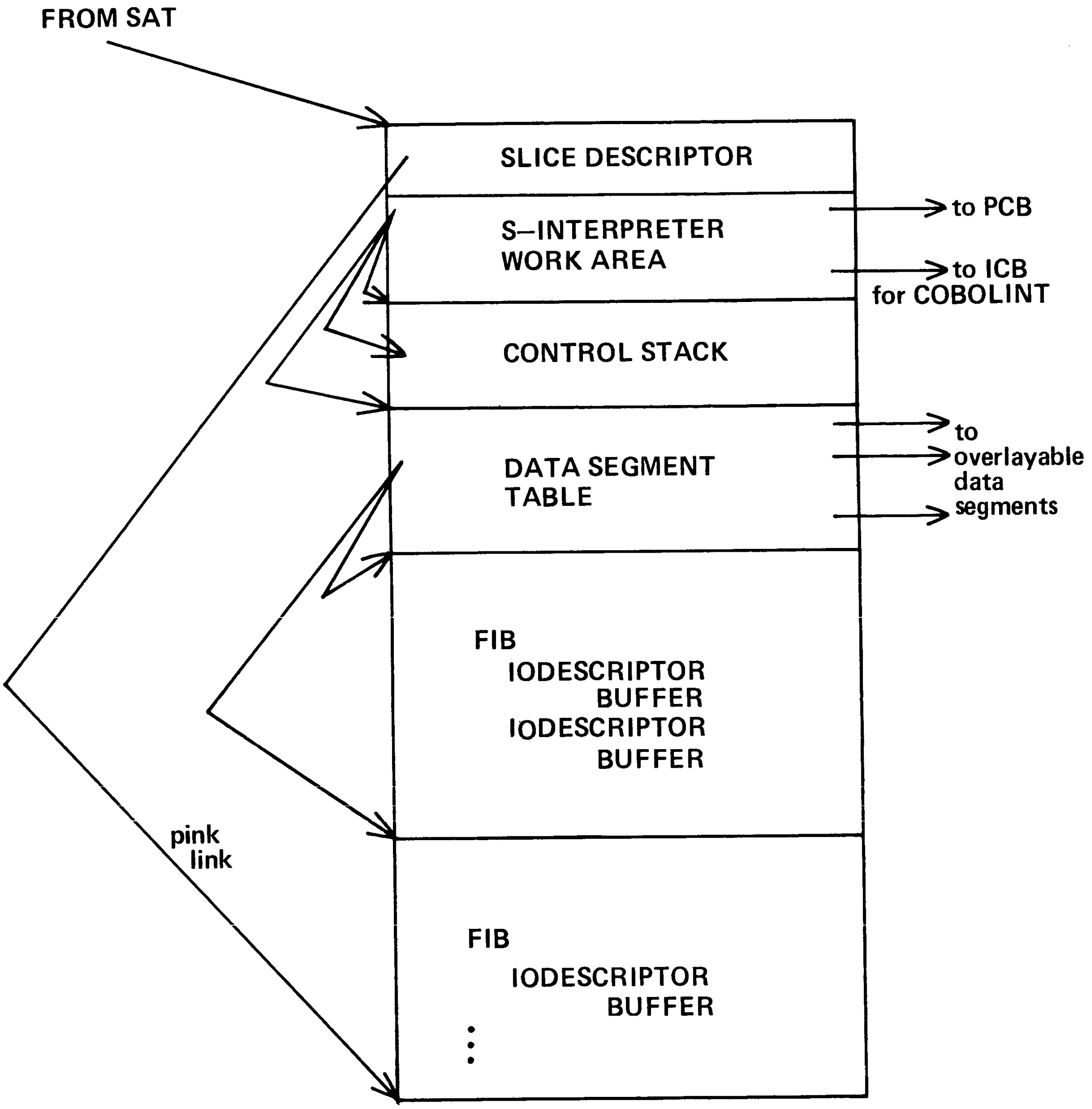
A logical file with the CMS indexed organisation consists of two physical files, a key file and a data file. The FIB of files with indexed organisation contains an IFIB which itself holds two buffers and I-O descriptors to the keyfile (see Table 8.7.1, IFIB MAP).

8.3.1 USER TASK COMPONENTS IN MEMORY



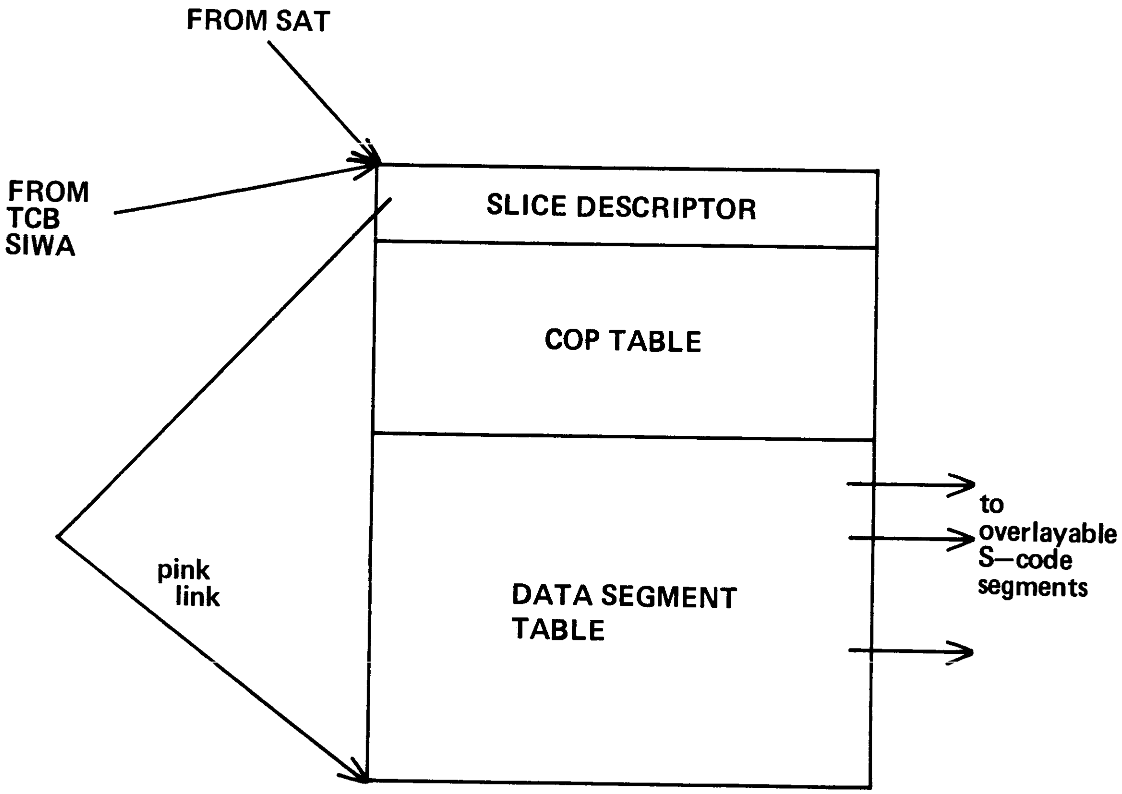
USER TASK COMPONENTS IN MEMORY

8.3.2 COBOL/RPG TCB COMPONENTS



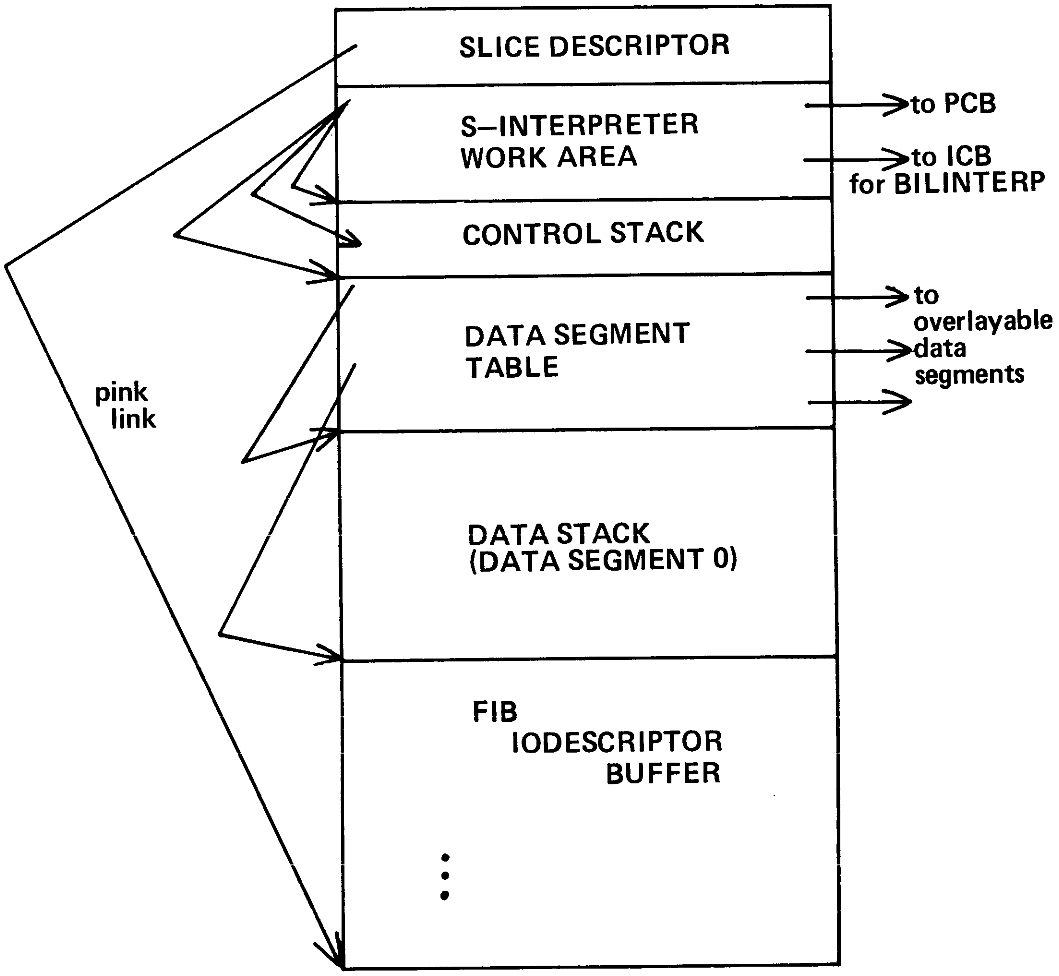
COBOL/RPG TCB COMPONENTS

8.3.3 COBOL/RPG PCB COMPONENTS



COBOL/RPG PCB COMPONENTS

8.3.4 MPLII (BIL) TCB COMPONENTS



MPLII (BIL) TCB COMPONENTS

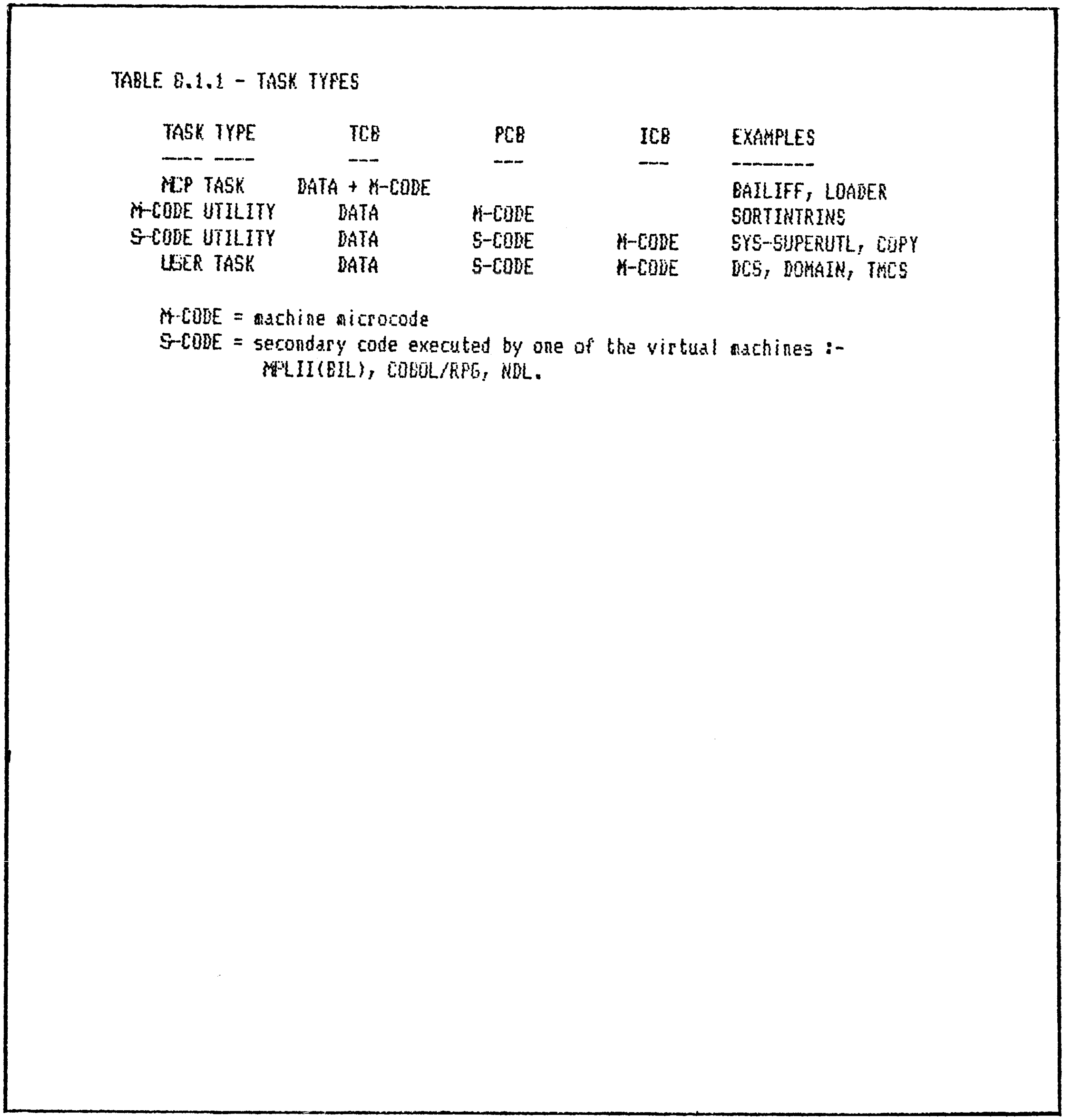


TABLE 8.1.1 - TASK TYPES

TASK TYPE	TCB	PCB	ICB	EXAMPLES
MCP TASK	DATA + M-CODE			BAILIFF, LOADER
M-CODE UTILITY	DATA	M-CODE		SORTINTRINS
S-CODE UTILITY	DATA	S-CODE	M-CODE	SYS-SUPERUTL, COPY
USER TASK	DATA	S-CODE	M-CODE	DCS, DOMAIN, TMCS

M-CODE = machine microcode
S-CODE = secondary code executed by one of the virtual machines :-
MPLII(BIL), COBOL/RPG, NDL.

TASK TYPES

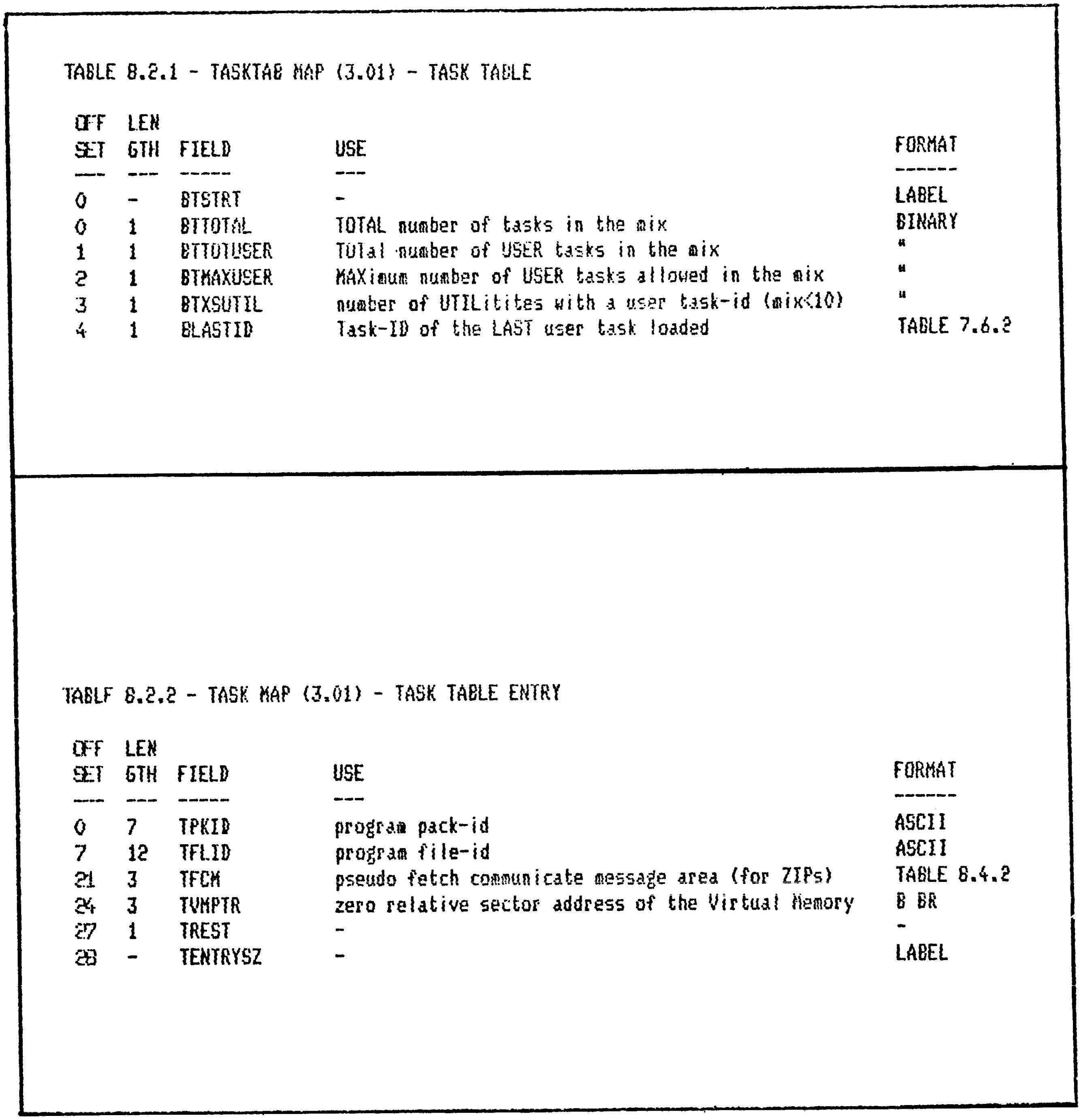


TABLE 8.2.1 - TASKTAB MAP (3.01) - TASK TABLE

OFF SET	LEN GTH	FIELD	USE	FORMAT
0	-	BTSTRT	-	LABEL
0	1	BTTOTAL	TOTAL number of tasks in the mix	BINARY
1	1	BTTOTUSER	TOtal number of USER tasks in the mix	"
2	1	BTMAXUSER	MAXimum number of USER tasks allowed in the mix	"
3	1	BTXSUTIL	number of UTILitites with a user task-id (mix<10)	"
4	1	BLASTID	Task-ID of the LAST user task loaded	TABLE 7.6.2

TABLF 8.2.2 - TASK MAP (3.01) - TASK TABLE ENTRY

OFF SET	LEN GTH	FIELD	USE	FORMAT
0	7	TPKID	program pack-id	ASCII
7	12	TFLID	program file-id	ASCII
21	3	TFCM	pseudo fetch communicate message area (for ZIPs)	TABLE 8.4.2
24	3	TVMPTR	zero relative sector address of the Virtual Memory	B BR
27	1	TREST	-	-
28	-	TENTRYSZ	-	LABEL

TASKTAB MAP (3.01) - TASK TABLE and
TASK MAP (3.01) - TASK TABLE ENTRY

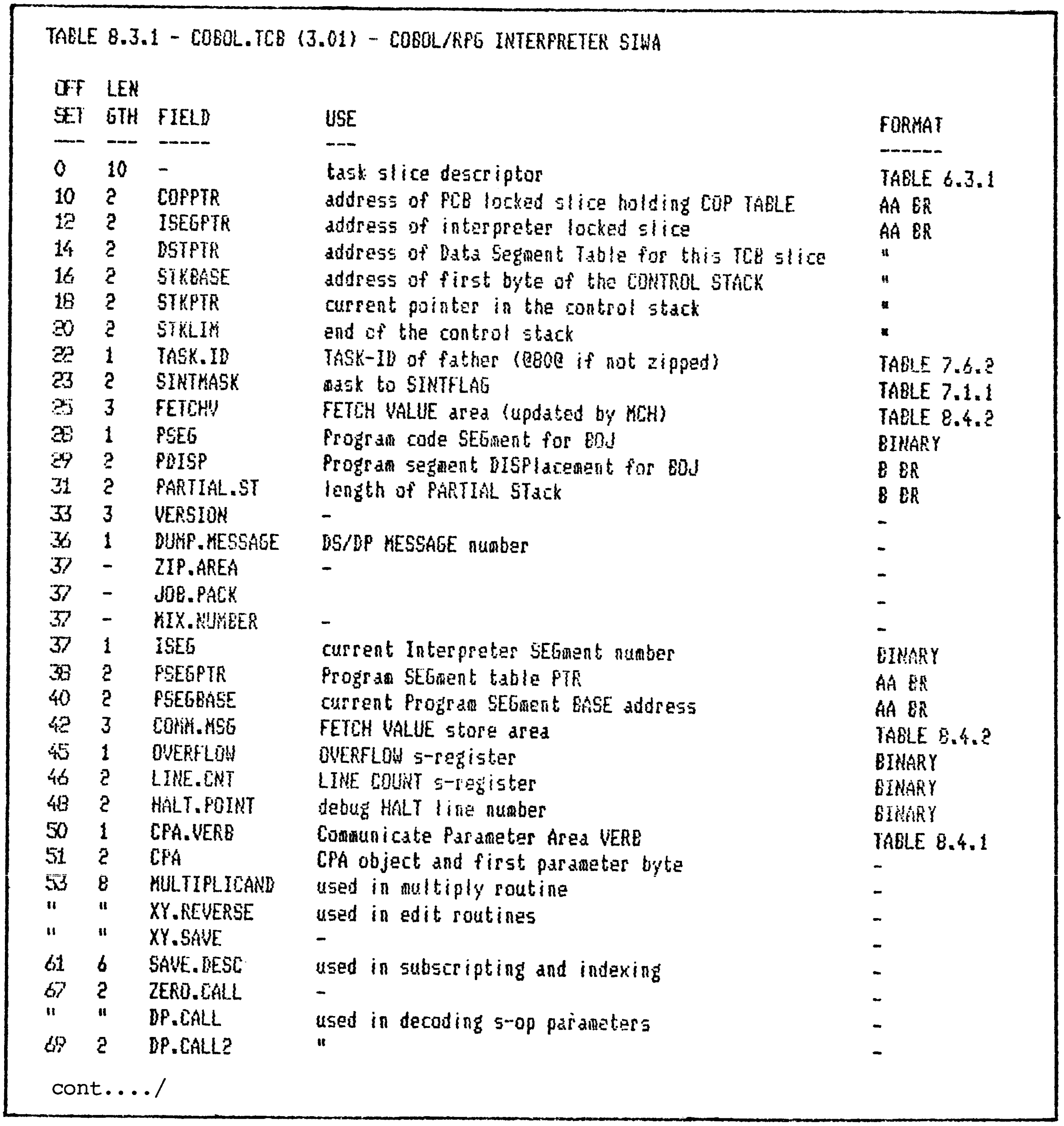


TABLE 8.3.1 - COBOL.TCB (3.01) - COBOL/RPG INTERPRETER SIWA

OFF	LEN			
SET	GTH	FIELD	USE	FORMAT
0	10	-	task slice descriptor	TABLE 6.3.1
10	2	COPPTR	address of PCB locked slice holding COP TABLE	AA BR
12	2	ISEGPTR	address of interpreter locked slice	AA BR
14	2	DSTPTR	address of Data Segment Table for this TCB slice	"
16	2	STKBASE	address of first byte of the CONTROL STACK	"
18	2	STKPTR	current pointer in the control stack	"
20	2	STKLIM	end of the control stack	"
22	1	TASK.ID	TASK-ID of father (@80@ if not zipped)	TABLE 7.6.2
23	2	SINTMASK	mask to SINTFLAG	TABLE 7.1.1
25	3	FETCHV	FETCH VALUE area (updated by MCH)	TABLE 8.4.2
28	1	PSEG	Program code SEGment for BOJ	BINARY
29	2	PDISP	Program segment DISPlacement for BOJ	B BR
31	2	PARTIAL.ST	length of PARTIAL STack	B BR
33	3	VERSION	-	-
36	1	DUMP.MESSAGE	DS/DP MESSAGE number	-
37	-	ZIP.AREA	-	-
37	-	JOB.PACK		-
37	-	MIX.NUMBER	-	-
37	1	ISEG	current Interpreter SEGment number	BINARY
38	2	PSEGPTR	Program SEGment table PTR	AA BR
40	2	PSEGBASE	current Program SEGment BASE address	AA BR
42	3	COMM.MSG	FETCH VALUE store area	TABLE 8.4.2
45	1	OVERFLOW	OVERFLOW s-register	BINARY
46	2	LINE.CNT	LINE COUNT s-register	BINARY
48	2	HALT.POINT	debug HALT line number	BINARY
50	1	CPA.VERB	Communicate Parameter Area VERB	TABLE 8.4.1
51	2	CPA	CPA object and first parameter byte	-
53	8	MULTIPLICAND	used in multiply routine	-
"	"	XY.REVERSE	used in edit routines	-
"	"	XY.SAVE	-	-
61	6	SAVE.DESC	used in subscripting and indexing	-
67	2	ZERO.CALL	-	-
"	"	DP.CALL	used in decoding s-op parameters	-
69	2	DP.CALL2	"	-

cont..../

COBOL.TCB (3.01) - COBOL/RPG INTERPRETER SIWA (cont.)

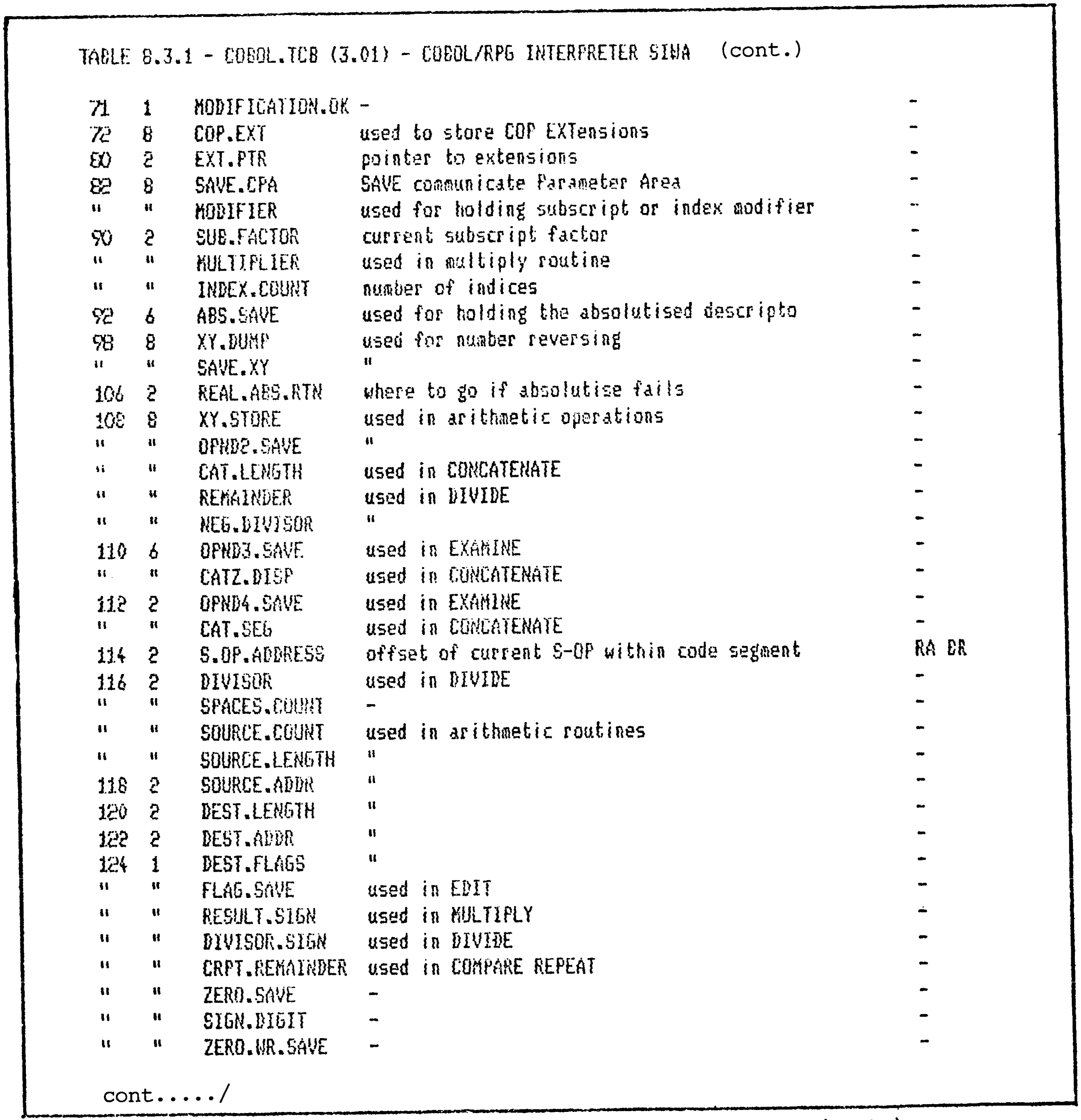


TABLE 8.3.1 - COBOL.TCB (3.01) - COBOL/RPG INTERPRETER SIWA (cont.)

71	1	MODIFICATION.OK	-	-
72	8	COP.EXT	used to store COP EXTensions	-
80	2	EXT.PTR	pointer to extensions	-
82	8	SAVE.CPA	SAVE communicate Parameter Area	-
"	"	MODIFIER	used for holding subscript or index modifier	-
90	2	SUB.FACTOR	current subscript factor	-
"	"	MULTIPLIER	used in multiply routine	-
"	"	INDEX.COUNT	number of indices	-
92	6	ABS.SAVE	used for holding the absolutised descripto	-
98	8	XY.DUMP	used for number reversing	-
"	"	SAVE.XY	"	-
106	2	REAL.ABS.RTN	where to go if absolutise fails	-
108	8	XY.STORE	used in arithmetic operations	-
"	"	OPND2.SAVE	"	-
"	"	CAT.LENGTH	used in CONCATENATE	-
"	"	REMAINDER	used in DIVIDE	-
"	"	NEG.DIVISOR	"	-
110	6	OPND3.SAVE	used in EXAMINE	-
"	"	CATZ.DISP	used in CONCATENATE	-
112	2	OPND4.SAVE	used in EXAMINE	-
"	"	CAT.SEG	used in CONCATENATE	-
114	2	S.OP.ADDRESS	offset of current S-OP within code segment	RA DR
116	2	DIVISOR	used in DIVIDE	-
"	"	SPACES.COUNT	-	-
"	"	SOURCE.COUNT	used in arithmetic routines	-
"	"	SOURCE.LENGTH	"	-
118	2	SOURCE.ADDR	"	-
120	2	DEST.LENGTH	"	-
122	2	DEST.ADDR	"	-
124	1	DEST.FLAGS	"	-
"	"	FLAG.SAVE	used in EDIT	-
"	"	RESULT.SIGN	used in MULTIPLY	-
"	"	DIVISOR.SIGN	used in DIVIDE	-
"	"	CRPT.REMAINDER	used in COMPARE REPEAT	-
"	"	ZERO.SAVE	-	-
"	"	SIGN.DIGIT	-	-
"	"	ZERO.WR.SAVE	-	-

cont...../

COBOL.TCB (3.01) - COBOL/RPG INTERPRETER SIWA (cont.)

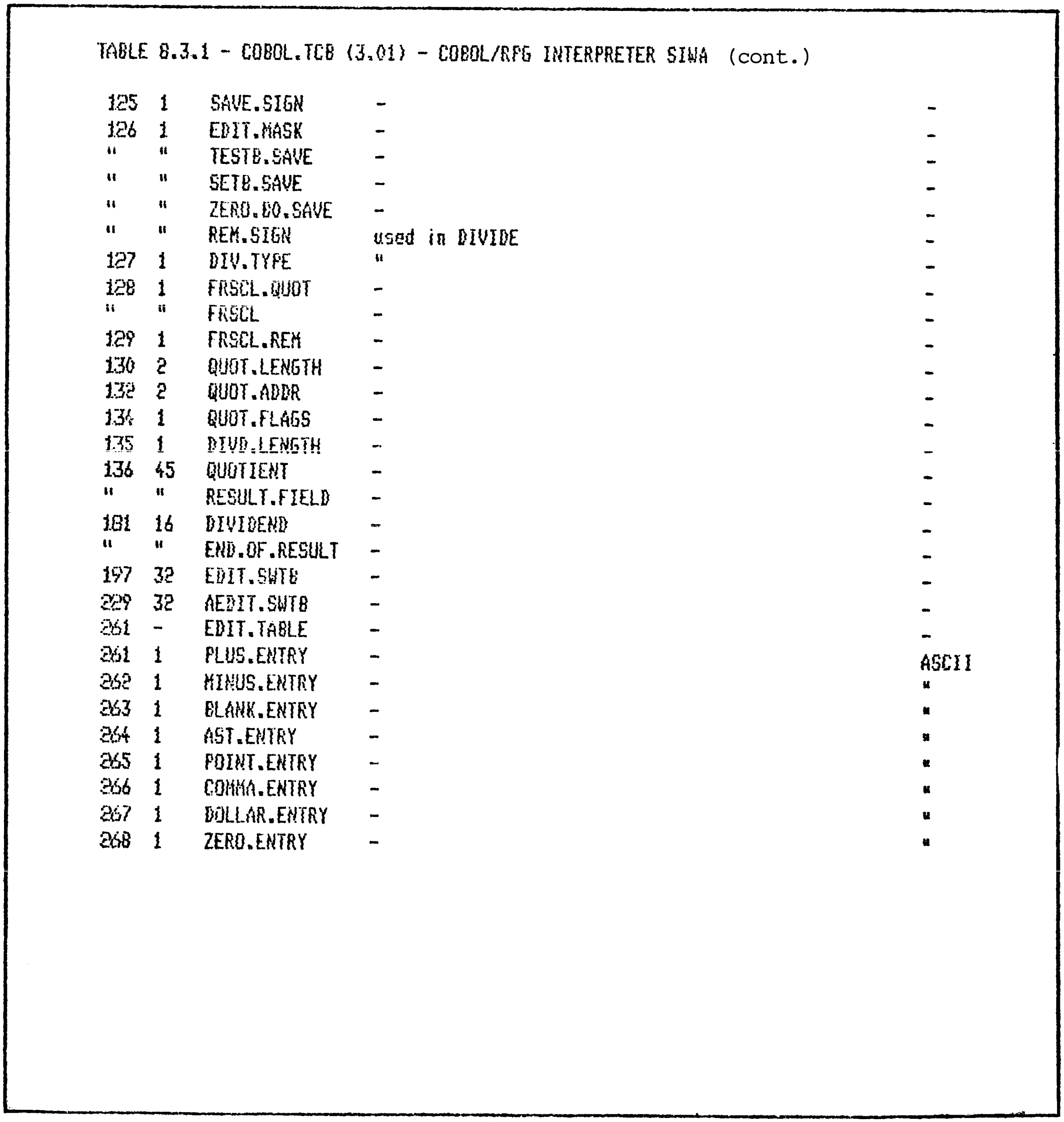


TABLE 8.3.1 - COBOL.TCB (3.01) - COBOL/RPG INTERPRETER SIWA (cont.)

125	1	SAVE.SIGN	-	-
126	1	EDIT.MASK	-	-
"	"	TESTB.SAVE	-	-
"	"	SETB.SAVE	-	-
"	"	ZERO.BO.SAVE	-	-
"	"	REM.SIGN	used in DIVIDE	-
127	1	DIV.TYPE	"	-
128	1	FRSCL.QUOT	-	-
"	"	FRSCL	-	-
129	1	FRSCL.REM	-	-
130	2	QUOT.LENGTH	-	-
132	2	QUOT.ADDR	-	-
134	1	QUOT.FLAGS	-	-
135	1	DIVD.LENGTH	-	-
136	45	QUOTIENT	-	-
"	"	RESULT.FIELD	-	-
181	16	DIVIDEND	-	-
"	"	END.OF.RESULT	-	-
197	32	EDIT.SWTB	-	-
229	32	AEDIT.SWTB	-	-
261	-	EDIT.TABLE	-	-
261	1	PLUS.ENTRY	-	ASCII
262	1	MINUS.ENTRY	-	"
263	1	BLANK.ENTRY	-	"
264	1	AST.ENTRY	-	"
265	1	POINT.ENTRY	-	"
266	1	COMMA.ENTRY	-	"
267	1	DOLLAR.ENTRY	-	"
268	1	ZERO.ENTRY	-	"

COBOL.TCB (3.01) - COBOL/RPG INTERPRETER SIWA (cont.)

TABLE 8.3.2 - MTCB MAP - MPLII (BIL) INTERPRETER WORK AREA

OFF SET	LEN GTH	FIELD	USE	FORMAT
0	10	ER.ADDR	task slice descriptor	TABLE 6.3.1
10	2	PSTA	Program Segment Table Address - code DST	AA BR
12	2	ISTA	Interpreter Segment Table Address	AA BR
14	2	DSTA	Data Segemnt Table Address	AA BR
16	2	CSB	Control Stack Base Address	AA BR
18	2	CSA	Control Stack Address (current pointer)	AA BR
20	2	CSL	Control Stack Limit address	AA BR
22	-	SIWA	-	LABEL
22	1	TASK.ID	TASK ID of the originator (@80@ if not ZIPped)	TABLE 7.6.2
23	2	SINTMASK	mask to SINTFLAG	TABLE 7.1.1
25	3	FETCHV	FETCH communicate Value returned by MCH	TABLE 8.4.2
28	3	SSA	S-code Start Address (segment + offset)	S+(AA BR)
31	2	P.STACK	Partial STACK length (user part of control stack)	B BR
33	1	ISEG	Interpreter SEGment number (=@00@ on 3.01)	SUBSCRIPT
34	2	CPA.PTR	work pointer in CPA	AA BR
36	2	CPA	Communicate Parameter Area	TABLE 8.4.1
36	1	CPA.VB	CPA VerB	"
37	1	CPA.OBJ	CPA OBJect (adverb)	-
38	22	CPA.REST	-	-
60	1	COMM.ERR	COMMunicate s-o_r (@63@=COMME,@73@=COMM)	-
61	2	CSEGB	current Code SEGment Base address	AA BR
63	1	ERR.NO	internal ERRror number	-
64	3	COMM.MESSAGE	fetch COMMunicate MESSAGE	TABLE 8.4.2
			updated by interpreter and accessed by FETCH.VALUE	-

cont...../

MTCB MAP - MPLII (BIL) INTERPRETER WORK AREA

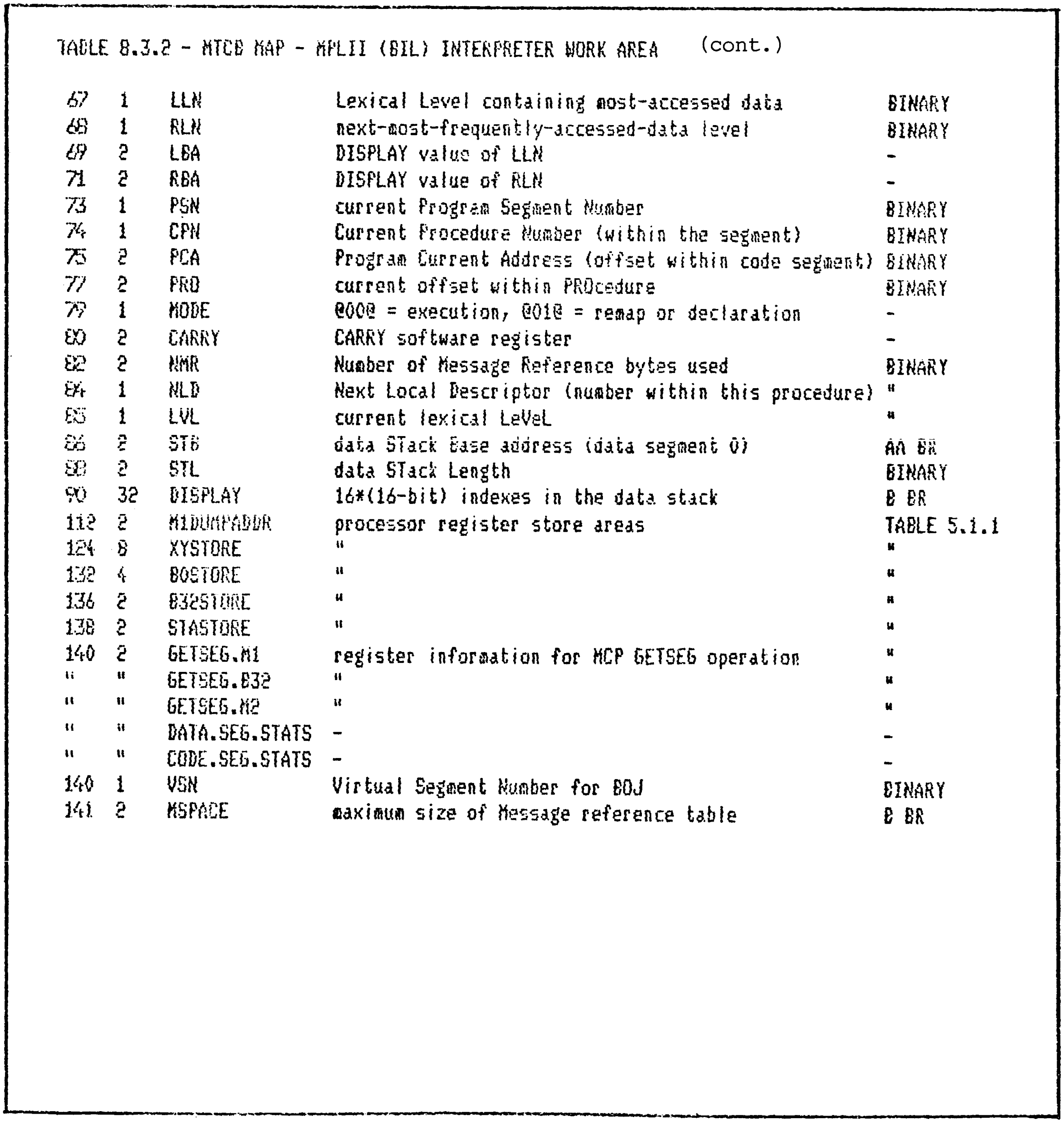


TABLE 8.3.2 - MTCB MAP - MPLII (BIL) INTERPRETER WORK AREA (cont.)

67	1	LLN	Lexical Level containing most-accessed data	BINARY
68	1	RLN	next-most-frequently-accessed-data level	BINARY
69	2	LBA	DISPLAY value of LLN	-
71	2	RBA	DISPLAY value of RLN	-
73	1	PSN	current Program Segment Number	BINARY
74	1	CPN	Current Procedure Number (within the segment)	BINARY
75	2	PCA	Program Current Address (offset within code segment)	BINARY
77	2	PRO	current offset within PROcedure	BINARY
79	1	MODE	@00@ = execution, @01@ = remap or declaration	-
80	2	CARRY	CARRY software register	-
82	2	NMR	Number of Message Reference bytes used	BINARY
84	1	NLD	Next Local Descriptor (number within this procedure)	"
85	1	LVL	current lexical LeVeL	"
86	2	STB	data STack Base address (data segment 0)	AA BR
88	2	STL	data STack Length	BINARY
90	32	DISPLAY	16*(16-bit) indexes in the data stack	B BR
112	2	M1DUMPADDR	processor register store areas	TABLE 5.1.1
124	8	XYSTORE	"	"
132	4	BOSTORE	"	"
136	2	B32STORE	"	"
138	2	STASTORE	"	"
140	2	GETSEG.M1	register information for MCP GETSEG operation	"
"	"	GETSEG.B32	"	"
"	"	GETSEG.M2	"	"
"	"	DATA.SEG.STATS	-	-
"	"	CODE.SEG.STATS	-	-
140	1	VSN	Virtual Segment Number for BOJ	BINARY
141	2	MSPACE	maximum size of Message reference table	B BR

MTCB MAP - MPLII (BIL) INTERPRETER WORK AREA (cont.)

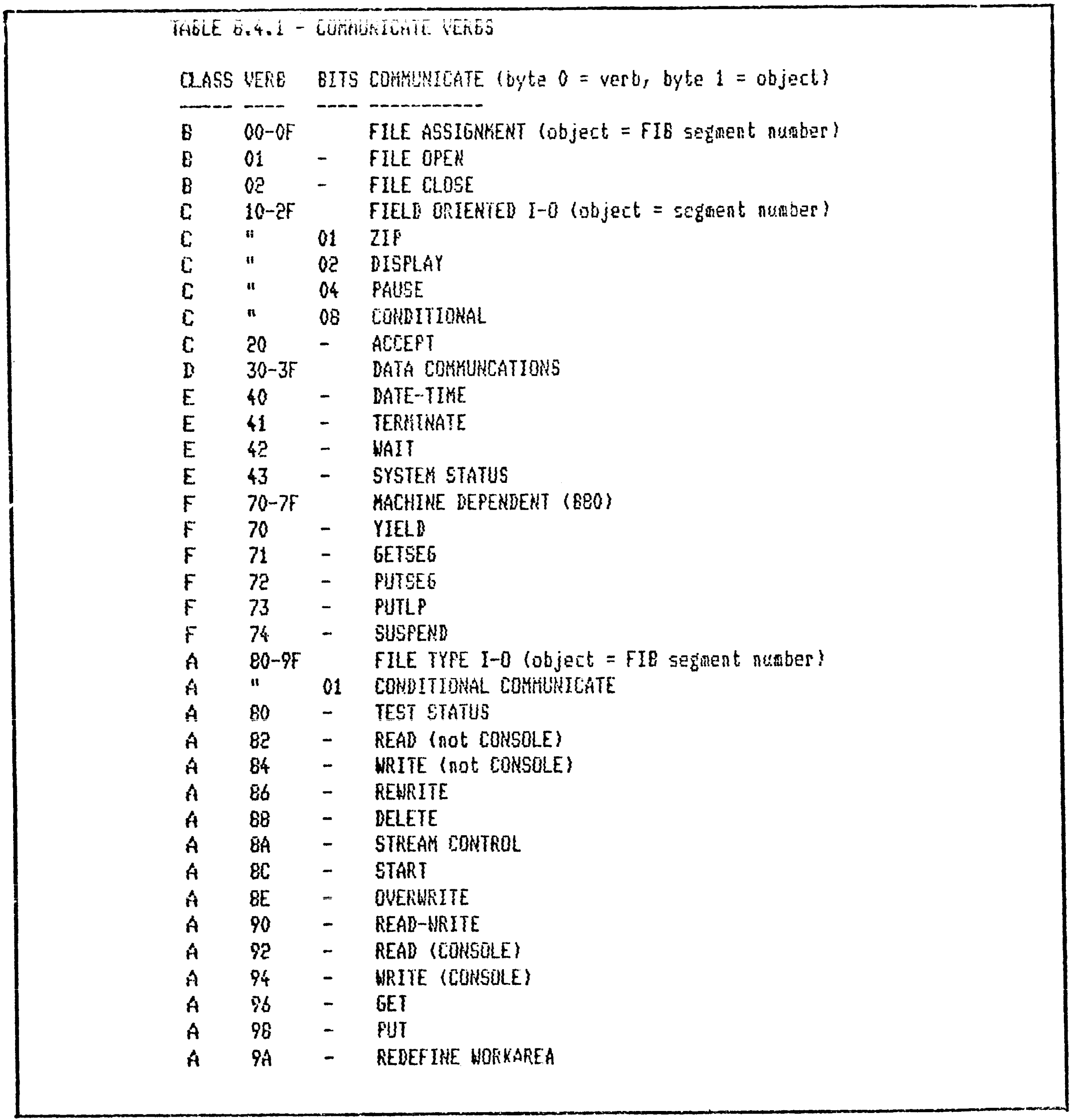


TABLE 8.4.1 - COMMUNICATE VERBS

CLASS	VERB	BITS	COMMUNICATE (byte 0 = verb, byte 1 = object)
B	00-0F		FILE ASSIGNMENT (object = FIB segment number)
B	01	-	FILE OPEN
B	02	-	FILE CLOSE
C	10-2F		FIELD ORIENTED I-O (object = segment number)
C	"	01	ZIP
C	"	02	DISPLAY
C	"	04	PAUSE
C	"	08	CONDITIONAL
C	20	-	ACCEPT
D	30-3F		DATA COMMUNCATIONS
E	40	-	DATE-TIME
E	41	-	TERMINATE
E	42	-	WAIT
E	43	-	SYSTEM STATUS
F	70-7F		MACHINE DEPENDENT (880)
F	70	-	YIELD
F	71	-	GETSEG
F	72	-	PUTSEG
F	73	-	PUTLP
F	74	-	SUSPEND
A	80-9F		FILE TYPE I-O (object = FIB segment number)
A	"	01	CONDITIONAL COMMUNICATE
A	80	-	TEST STATUS
A	82	-	READ (not CONSOLE)
A	84	-	WRITE (not CONSOLE)
A	86	-	REWRITE
A	88	-	DELETE
A	8A	-	STREAM CONTROL
A	8C	-	START
A	8E	-	OVERWRITE
A	90	-	READ-WRITE
A	92	-	READ (CONSOLE)
A	94	-	WRITE (CONSOLE)
A	96	-	GET
A	98	-	PUT
A	9A	-	REDEFINE WORKAREA

COMMUNICATE VERBS

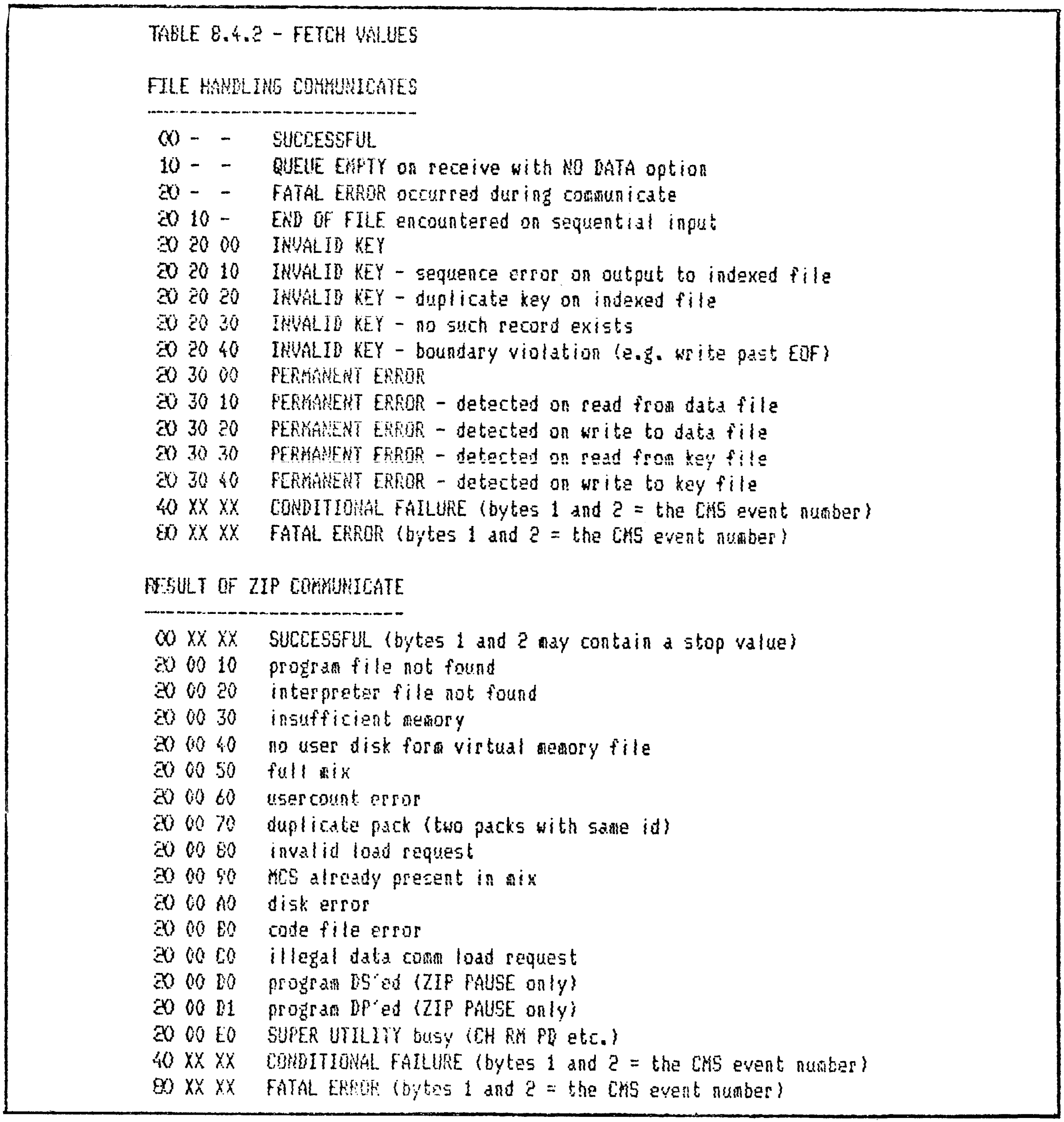


TABLE 8.4.2 - FETCH VALUES

FILE HANDLING COMMUNICATES

00 - -	SUCCESSFUL
10 - -	QUEUE EMPTY on receive with NO DATA option
20 - -	FATAL ERROR occurred during communicate
20 10 -	END OF FILE encountered on sequential input
20 20 00	INVALID KEY
20 20 10	INVALID KEY - sequence error on output to indexed file
20 20 20	INVALID KEY - duplicate key on indexed file
20 20 30	INVALID KEY - no such record exists
20 20 40	INVALID KEY - boundary violation (e.g. write past EOF)
20 30 00	PERMANENT ERROR
20 30 10	PERMANENT ERROR - detected on read from data file
20 30 20	PERMANENT ERROR - detected on write to data file
20 30 30	PERMANENT ERROR - detected on read from key file
20 30 40	PERMANENT ERROR - detected on write to key file
40 XX XX	CONDITIONAL FAILURE (bytes 1 and 2 = the CMS event number)
80 XX XX	FATAL ERROR (bytes 1 and 2 = the CMS event number)

RESULT OF ZIP COMMUNICATE

00 XX XX	SUCCESSFUL (bytes 1 and 2 may contain a stop value)
20 00 10	program file not found
20 00 20	interpreter file not found
20 00 30	insufficient memory
20 00 40	no user disk form virtual memory file
20 00 50	full mix
20 00 60	usercount error
20 00 70	duplicate pack (two packs with same id)
20 00 80	invalid load request
20 00 90	MCS already present in mix
20 00 A0	disk error
20 00 B0	code file error
20 00 C0	illegal data comm load request
20 00 D0	program DS'ed (ZIP PAUSE only)
20 00 D1	program DP'ed (ZIP PAUSE only)
20 00 E0	SUPER UTILITY busy (CH RM PD etc.)
40 XX XX	CONDITIONAL FAILURE (bytes 1 and 2 = the CMS event number)
80 XX XX	FATAL ERROR (bytes 1 and 2 = the CMS event number)

FETCH VALUES (cont.)

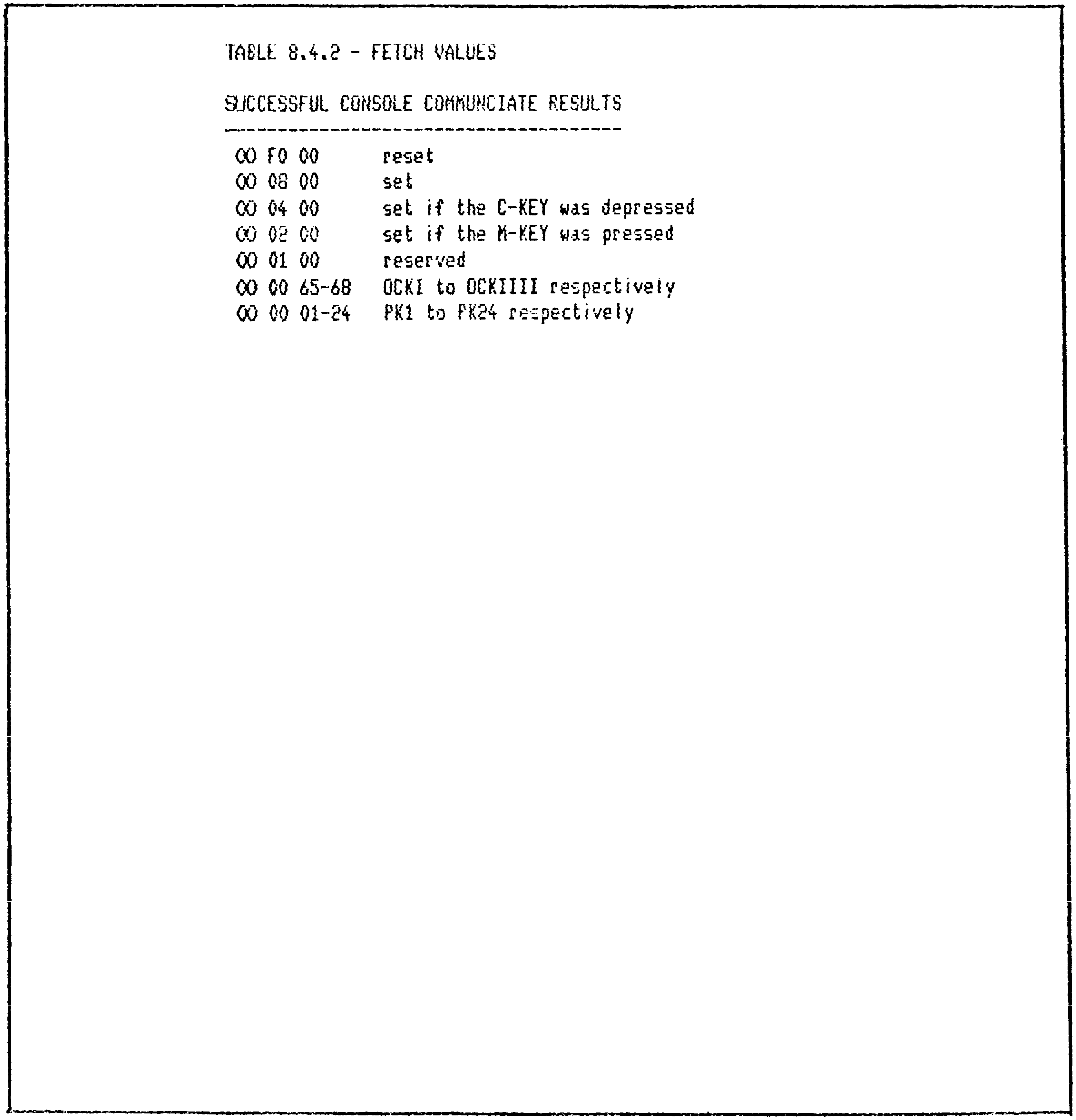


TABLE 8.4.2 - FETCH VALUES

SUCCESSFUL CONSOLE COMMUNCIATE RESULTS

00 F0 00	reset
00 08 00	set
00 04 00	set if the C-KEY was depressed
00 02 00	set if the M-KEY was pressed
00 01 00	reserved
00 00 65-68	OCKI to OCKIIII respectively
00 00 01-24	PK1 to PK24 respectively

FETCH VALUES

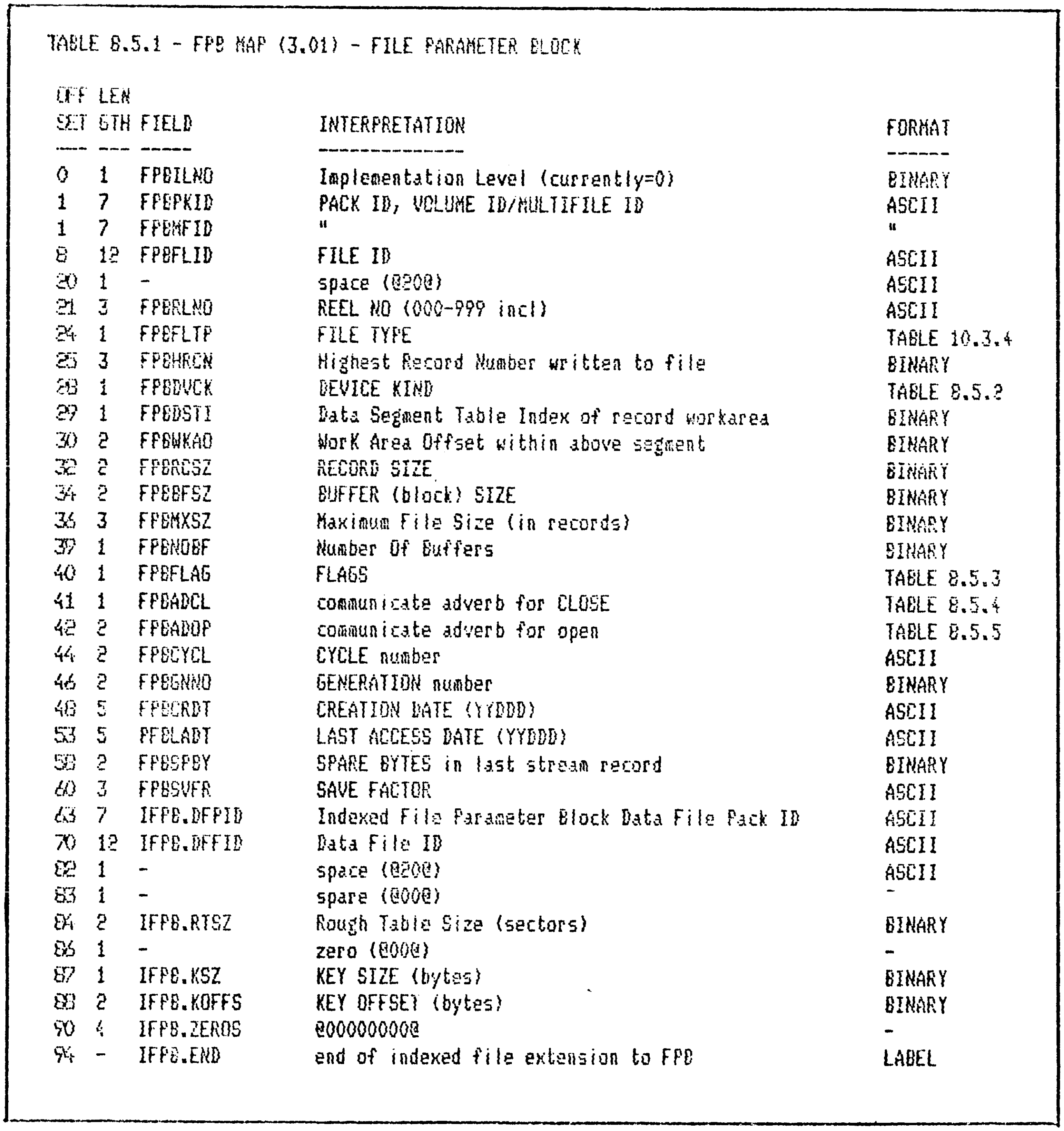


TABLE 8.5.1 - FPB MAP (3.01) - FILE PARAMETER BLOCK

OFF SET	LEN GTH	FIELD	INTERPRETATION	FORMAT
0	1	FPBILNO	Implementation Level (currently=0)	BINARY
1	7	FPBPKID	PACK ID, VOLUME ID/MULTIFILE ID	ASCII
1	7	FPBMFID	"	"
8	12	FPBFLID	FILE ID	ASCII
20	1	-	space (@20@)	ASCII
21	3	FPBRLNO	REEL NO (000-999 incl)	ASCII
24	1	FPBFLTP	FILE TYPE	TABLE 10.3.4
25	3	FPBHRCN	Highest Record Number written to file	BINARY
28	1	FPBDVCK	DEVICE KIND	TABLE 8.5.2
29	1	FPBDSTI	Data Segment Table Index of record workarea	BINARY
30	2	FPBWKAO	WorK Area Offset within above segment	BINARY
32	2	FPBRCSZ	RECORD SIZE	BINARY
34	2	FPBBFSZ	BUFFER (block) SIZE	BINARY
36	3	FPBMXSZ	Maximum File Size (in records)	BINARY
39	1	FPBNOBF	Number Of Buffers	BINARY
40	1	FPBFLAG	FLAGS	TABLE 8.5.3
41	1	FPBADCL	communicate adverb for CLOSE	TABLE 8.5.4
42	2	FPBADOP	communicate adverb for open	TABLE 8.5.5
44	2	FPBCYCL	CYCLE number	ASCII
46	2	FPBGNNO	GENERATION number	BINARY
48	5	FPBCRDT	CREATION DATE (YYDDD)	ASCII
53	5	PFBLADT	LAST ACCESS DATE (YYDDD)	ASCII
58	2	FPBSPBY	SPARE BYTES in last stream record	BINARY
60	3	FPBSVFR	SAVE FACTOR	ASCII
63	7	IFPB.DFPID	Indexed File Parameter Block Data File Pack ID	ASCII
70	12	IFPB.DFFID	Data File ID	ASCII
82	1	-	space (@20@)	ASCII
83	1	-	spare (@00@)	-
84	2	IFPB.RTSZ	Rough Table Size (sectors)	BINARY
86	1	-	zero (@00@)	-
87	1	IFPB.KSZ	KEY SIZE (bytes)	BINARY
88	2	IFPB.KOFFS	KEY OFFSET (bytes)	BINARY
90	4	IFPB.ZEROS	@00000000@	-
94	-	IFPB.END	end of indexed file extension to FPB	LABEL

FPB MAP (3.01) - FILE PARAMETER BLOCK

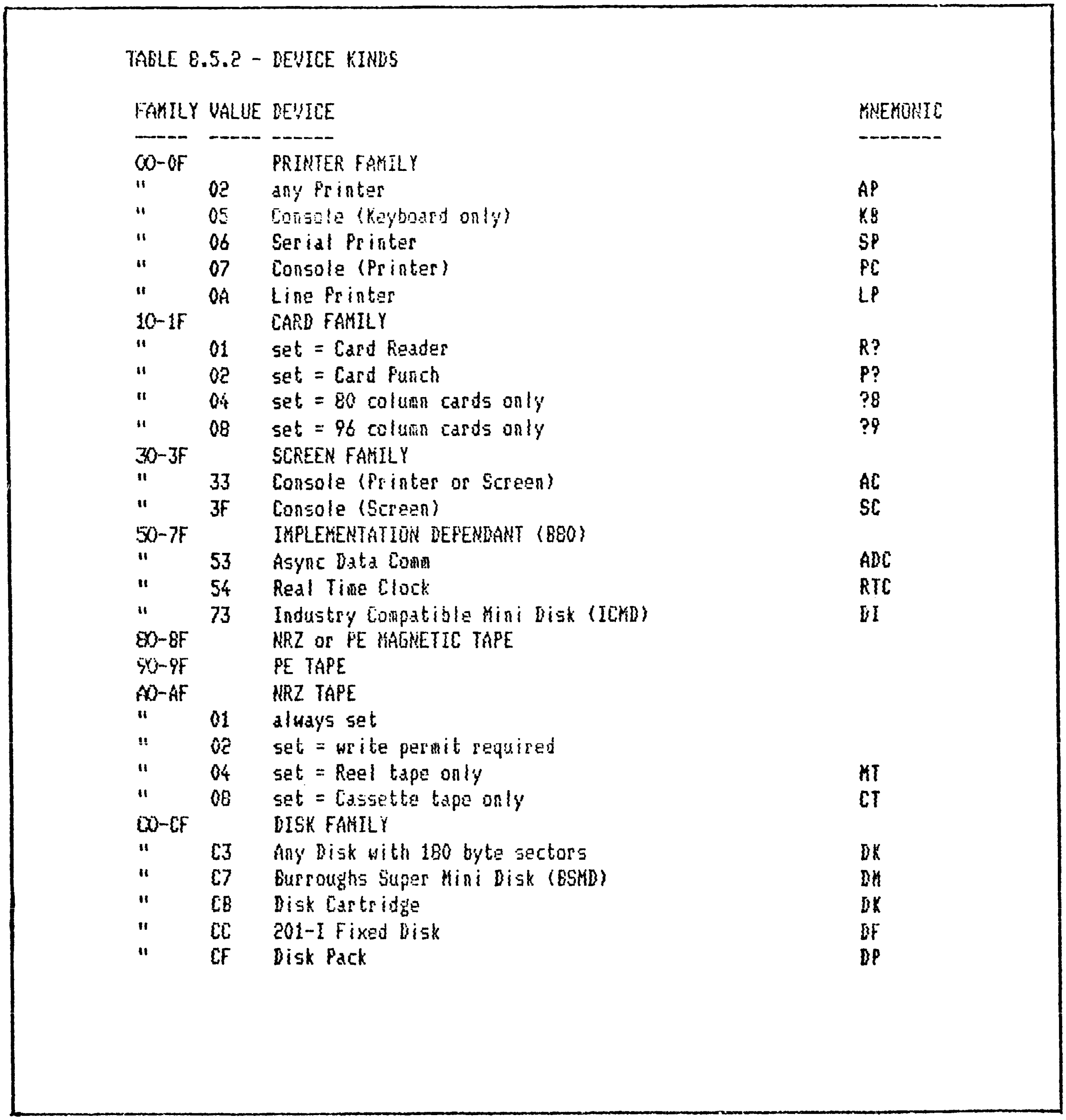


TABLE 8.5.2 - DEVICE KINDS

FAMILY	VALUE	DEVICE	MNEMONIC
00-0F		PRINTER FAMILY	
"	02	any Printer	AP
"	05	Console (Keyboard only)	KB
"	06	Serial Printer	SP
"	07	Console (Printer)	PC
"	0A	Line Printer	LP
10-1F		CARD FAMILY	
"	01	set = Card Reader	R?
"	02	set = Card Punch	P?
"	04	set = 80 column cards only	?8
"	08	set = 96 column cards only	?9
30-3F		SCREEN FAMILY	
"	33	Console (Printer or Screen)	AC
"	3F	Console (Screen)	SC
50-7F		IMPLEMENTATION DEPENDANT (B80)	
"	53	Async Data Comm	ADC
"	54	Real Time Clock	RTC
"	73	Industry Compatible Mini Disk (ICMD)	DI
80-8F		NRZ or PE MAGNETIC TAPE	
90-9F		PE TAPE	
A0-AF		NRZ TAPE	
"	01	always set	
"	02	set = write permit required	
"	04	set = Reel tape only	MT
"	08	set = Cassette tape only	CT
C0-CF		DISK FAMILY	
"	C3	Any Disk with 180 byte sectors	DK
"	C7	Burroughs Super Mini Disk (BSMD)	DM
"	C8	Disk Cartridge	DK
"	CC	201-I Fixed Disk	DF
"	CF	Disk Pack	DP

DEVICE KINDS

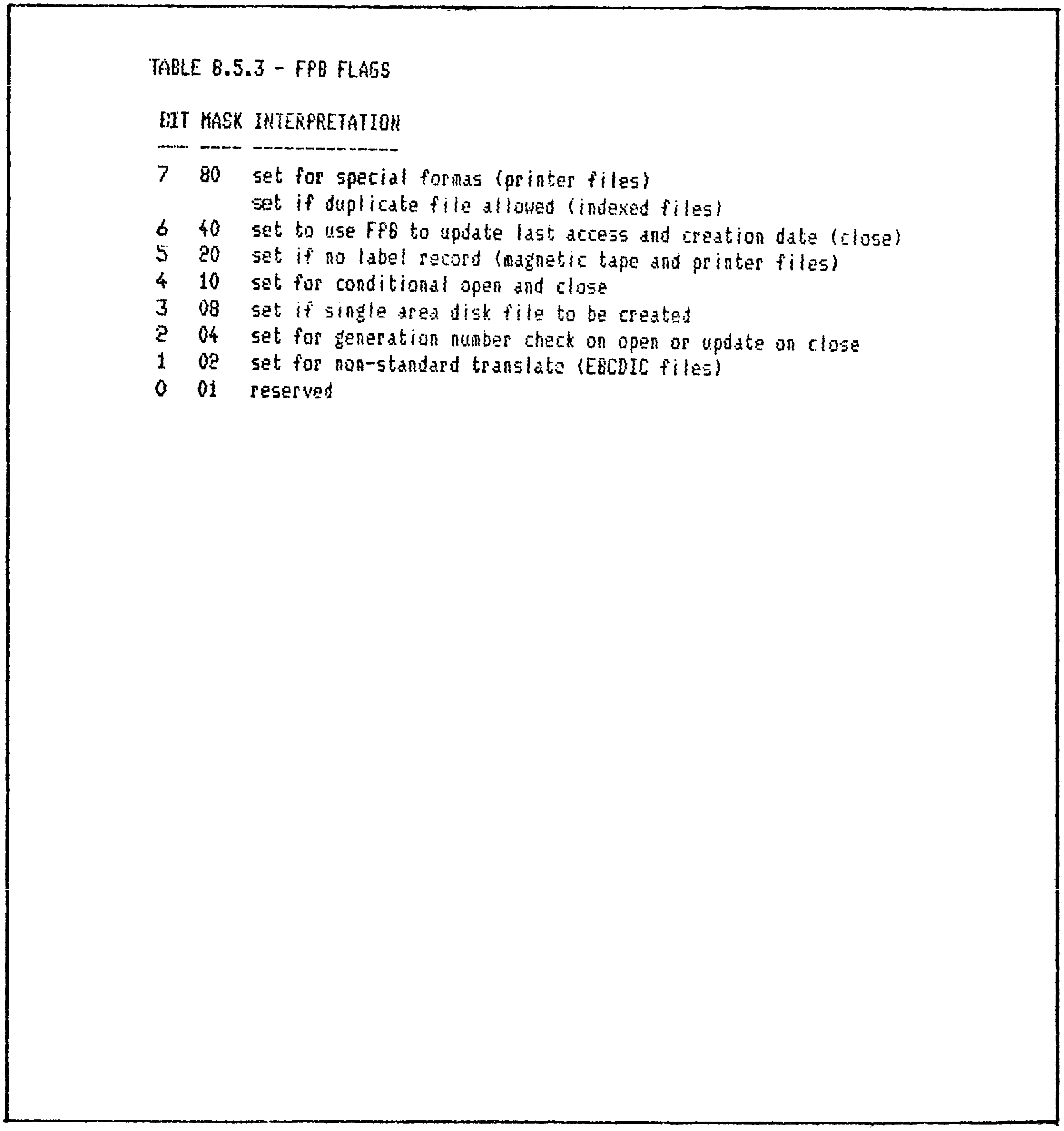


TABLE 8.5.3 - FPB FLAGS

BIT	MASK	INTERPRETATION
7	80	set for special formas (printer files)
		set if duplicate file allowed (indexed files)
6	40	set to use FPB to update last access and creation date (close)
5	20	set if no label record (magnetic tape and printer files)
4	10	set for conditional open and close
3	08	set if single area disk file to be created
2	04	set for generation number check on open or update on close
1	02	set for non-standard translate (EBCDIC files)
0	01	reserved

FPB FLAGS

TABLE 8.5.4 - ADVERB FOR OPEN

BITS	MASK	VALUE	INTERPRETATION
15	8000		reserved
14	4000	1	open EXTEND
13-12	3000		OTHERUSE
"	"	11,10	free access (normal)
"	"	01	lock access (other users may only read)
"	"	00	lock (no other users allowed)
11-10	0C00		MYUSE
"	"	10	output
"	"	01	input
"	"	11	input-output
9-6	03C0		reserved
5-4	0030		ACCESSMODE
"	"	00	illegal
"	"	01	random
"	"	10	sequential
"	"	11	stream
3-0	000F		reserved

TABLE 8.5.5 - ADVERB FOR CLOSE

BITS	MASK	VALUE	INTERPRETATION
7	80	-	reserved
6	40	1	close with no rewind
"	"	0	change reel leaving file open
5-3	38	000	half close
"	"	011	close with lock
"	"	101	close with purge
"	"	111	close with remove
"	"	001	close with release
"	"	-	all other combinations treated as half-close
2	04	1	crunch
1	02	1	merge overflow region into index region of indexed file
0	01	-	reserved

ADVERB FOR OPEN and
ADVERB FOR CLOSE

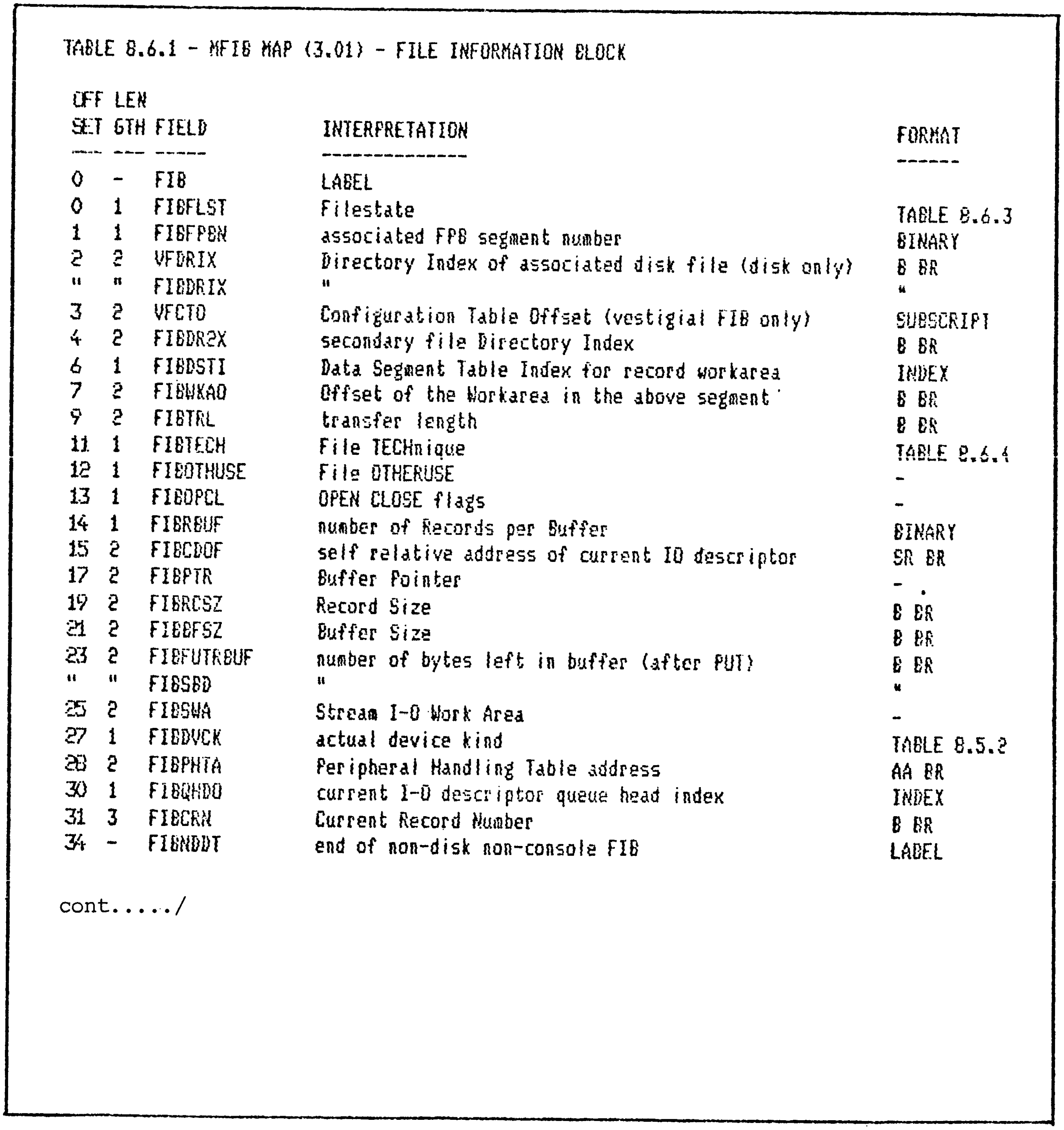


TABLE 8.6.1 - MFIB MAP (3.01) - FILE INFORMATION BLOCK

OFF SET	LEN GTH	FIELD	INTERPRETATION	FORMAT
0	-	FIB	LABEL	
0	1	FIBFLST	Filestate	TABLE 8.6.3
1	1	FIBFPBN	associated FPB segment number	BINARY
2	2	VFDRIX	Directory Index of associated disk file (disk only)	B BR
"	"	FIBDRIX	"	"
3	2	VFCTO	Configuration Table Offset (vestigial FIB only)	SUBSCRIPT
4	2	FIBDR2X	secondary file Directory Index	B BR
6	1	FIBDSTI	Data Segment Table Index for record workarea	INDEX
7	2	FIBWKAO	Offset of the Workarea in the above segment'	B BR
9	2	FIBTRL	transfer length	B BR
11	1	FIBTECH	File TECHnique	TABLE 8.6.4
12	1	FIBOTHUSE	File OTHERUSE	-
13	1	FIBOPCL	OPEN CLOSE flags	-
14	1	FIBRBUF	number of Records per Buffer	BINARY
15	2	FIBCDOF	self relative address of current IO descriptor	SR BR
17	2	FIBPTR	Buffer Pointer	-
19	2	FIBRCSZ	Record Size	B BR
21	2	FIBBFSZ	Buffer Size	B BR
23	2	FIBFUTRBUF	number of bytes left in buffer (after PUT)	B BR
"	"	FIBSBD	"	"
25	2	FIBSWA	Stream I-O Work Area	-
27	1	FIBDVCK	actual device kind	TABLE 8.5.2
28	2	FIBPHTA	Peripheral Handling Table address	AA BR
30	1	FIBQHDO	current I-O descriptor queue head index	INDEX
31	3	FIBCRN	Current Record Number	B BR
34	-	FIBNDDT	end of non-disk non-console FIB	LABEL

cont...../

MFIB_MAP (3.01) - FILE INFORMATION BLOCK

TABLE 8.6.1 - MFIB MAP (3.01) - FILE INFORMATION BLOCK (cont.)

34	2	FIBCONSIZE	(CONS) number of columns on a console device	B BR
34	3	FIBNRR	(DISK) Next Record to be Read	B BR
36	1	FIBHDPOS	(CONS) column position of print head	BINARY
37	3	FIBLDKA	(DISK) last Disk record Accessed	B BR
38	2	FIBLOGICALHDPOS	(CONS) Logical print Head Position	B BR
40	16	FIBINDICS	(CONS) Keyboard Indicators (lights)	-
40	2	FIBSEC	(DISK) Buffer length in SECtors	B BR
42	3	FIBMRAP	(DISK) Maximium Record Number in current Area	B BR
45	3	FIBMRW	(DISK) Maximum Record Written	B BR
48	3	FIBMRD	(DISK) Maximum Record Declared	B BR
51	1	FIBARN	(DISK) Number of Areas allocated	BINARY
52	1	FIBPRUN	(DISK) Primary Disk Unit Number (@FF@=unallocated)	-
53	1	FIBSCUN	(DISK) Secondary disk Unit Number (@FF@=unallocated)	-
54	2	FIBILINK	(DISK) link to indexed file flags (IKEYFLAG)	SR BR
56	2	FIBTAB	(CONS) TAB column number	B BR
56	-	FIBDFH	(DISK) memory Disk File Header	LABEL
56	2	FIBLA1	(DISK) number of sectors in disk area 1	B BR
58	3	FIBDA1	(DISK) sector address of disk area 1	B BR
61	75	-	(DISK) 15 more length-address pairs	-
136	-	FIBEND	end of FIB for DISK files	LABEL

MFIB MAP (3.01) - FILE INFORMATION BLOCK

TABLE 8.6.2 - MAP VEST.FIB (3.01) - VESTIGIAL FIB

OFF SET	LEN GTH	FIELD	INTERPRETATION	FORMAT
0	1	VEST.FIB.FLAGS	segment descriptor flags (=0480)	TABLE 6.4.2
1	2	VEST.FIB.BASE	segment base (points to filestate below)	AA BR
3	1	VEST.FIB.FS	filestate	TABLE 8.6.3
4	1	VEST.FIB.FPB	associated FPB segment number	BINARY
5	2	VEST.FIB.DRX	(DISK) Directory Index of File (2 8-bit values)	BINARY
6	2	VEST.FIB.OFFCT	(NON-DISK) Offset of Configuration Table	RA BR
7	1	VEST.FIB.DU	(DISK) Disk Unit	-

TABLE 8.6.3 - FIB FILESTATE

BITS	MASK	VALUE	INTERPRETATION
7	80	80	Sequential Oraganisation (not Indexed)
6-5	60		I-O MASK
"	40	40	Output allowed
"	20	20	Input allowed
4-3	18		ACCESS TECHNIQUE
"		00	Stream
"		08	Console
"		10	Random
"		18	Sequential
2	04	04	End Of File not reached
1	02	02	File Half Closed
0	01	00	Stream

MAP VEST.FIB (3.01) - VESTIGIAL FIB and FIB FILESTATE

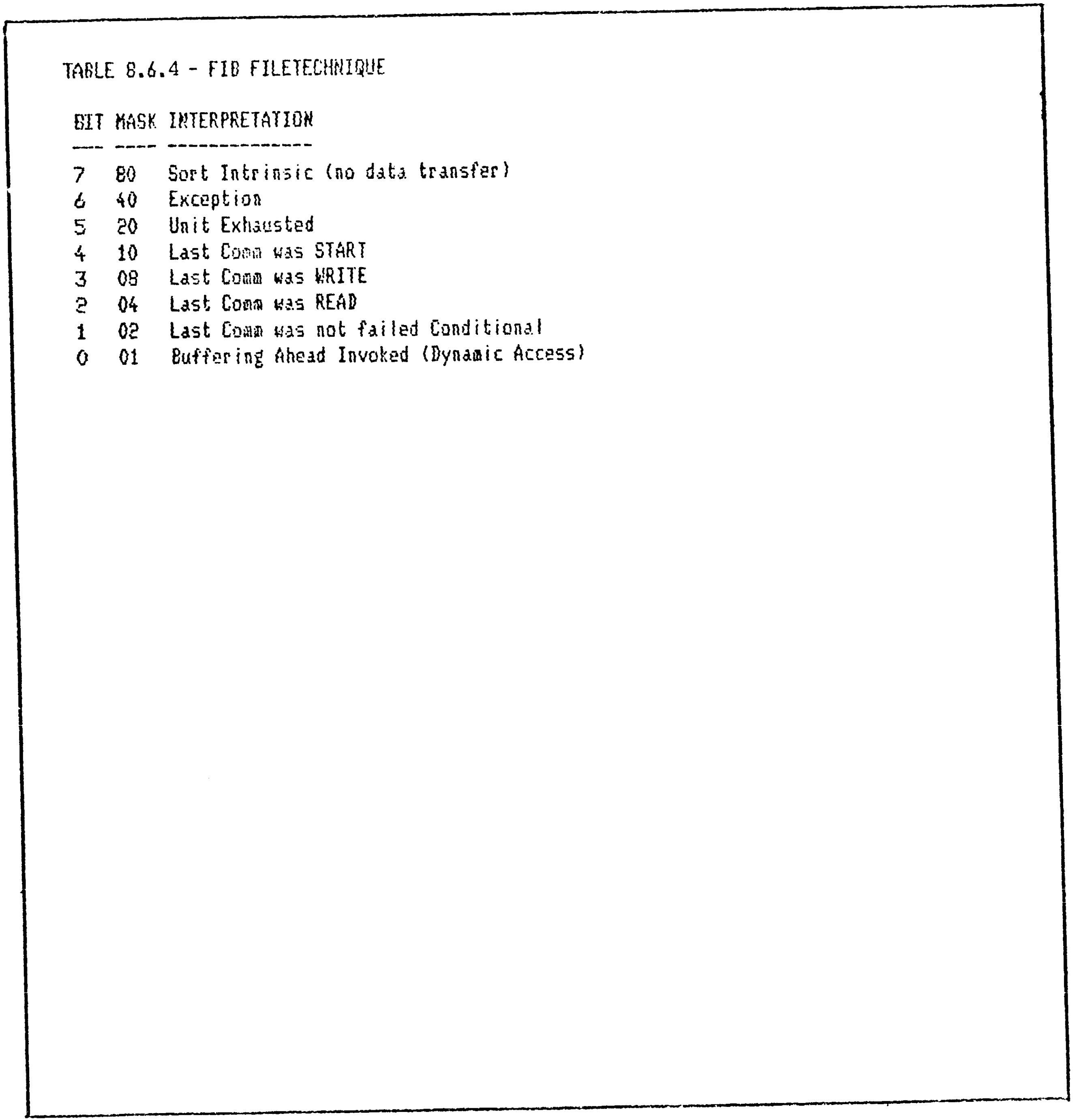


TABLE 8.6.4 - FIB FILETECHNIQUE

BIT	MASK	INTERPRETATION
7	80	Sort Intrinsic (no data transfer)
6	40	Exception
5	20	Unit Exhausted
4	10	Last Comm was START
3	08	Last Comm was WRITE
2	04	Last Comm was READ
1	02	Last Comm was not failed Conditional
0	01	Buffering Ahead Invoked (Dynamic Access)

FIB FILETECHNIQUE

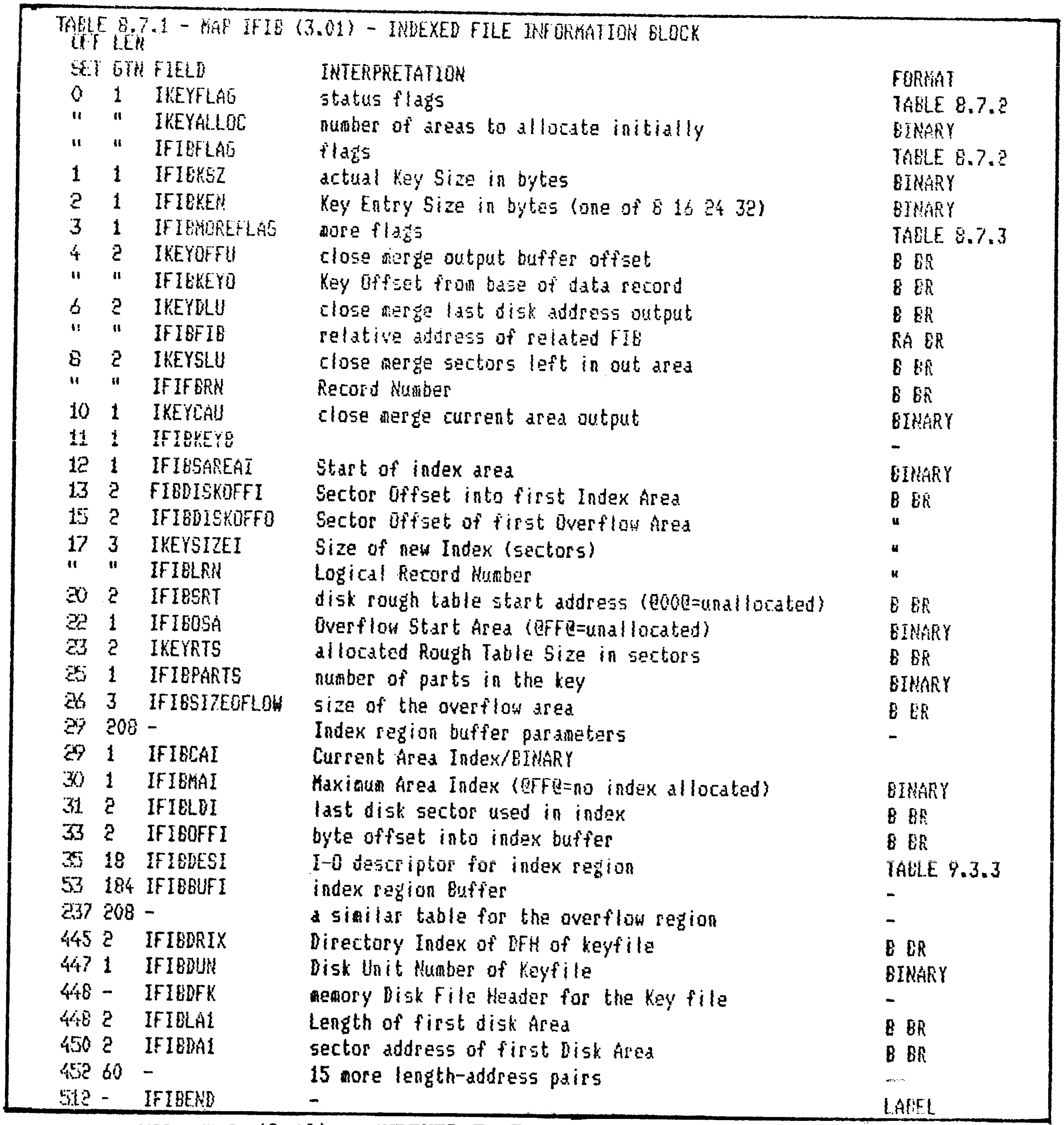


TABLE 8.7.1 - MAP IFIB (3.01) - INDEXED FILE INFORMATION BLOCK

OFF	LEN			
SET	GTN	FIELD	INTERPRETATION	FORMAT
0	1	IKEYFLAG	status flags	TABLE 8.7.2
"	"	IKEYALLOC	number of areas to allocate initially	BINARY
"	"	IFIBFLAG	flags	TABLE 8.7.2
1	1	IFIBKSZ	actual Key Size in bytes	BINARY
2	1	IFIBKEN	Key Entry Size in bytes (one of 8 16 24 32)	BINARY
3	1	IFIBMOREFLAG	more flags	TABLE 8.7.3
4	2	IKEYOFFU	close merge output buffer offset	B BR
"	"	IFIBKEYO	Key Offset from base of data record	B BR
6	2	IKEYDLU	close merge last disk address output	B BR
"	"	IFIBFIB	relative address of related FIB	RA BR
8	2	IKEYSLU	close merge sectors left in out area	B BR
"	"	IFIFBRN	Record Number	B BR
10	1	IKEYCAU	close merge current area output	BINARY
11	1	IFIBKEYB		-
12	1	IFIBSAREAI	Start of index area	BINARY
13	2	FIBDISKOFFI	Sector Offset into first Index Area	B BR
15	2	IFIBDISKOFFO	Sector Offset of first Overflow Area	"
17	3	IKEYSIZEI	Size of new Index (sectors)	"
"	"	IFIBLRN	Logical Record Number	"
20	2	IFIBSRT	disk rough table start address (@00@=unallocated)	B BR
22	1	IFIBOSA	Overflow Start Area (@FF@=unallocated)	BINARY
23	2	IKEYRTS	allocated Rough Table Size in sectors	B BR
25	1	IFIBPARTS	number of parts in the key	BINARY
26	3	IFIBSIZEOFLOW	size of the overflow area	B BR
29	208	-	Index region buffer parameters	-
29	1	IFIBCAI	Current Area Index/BINARY	
30	1	IFIBMAI	Maximum Area Index (@FF@=no index allocated)	BINARY
31	2	IFIBLDI	last disk sector used in index	B BR
33	2	IFIBOFFI	byte offset into index buffer	B BR
35	18	IFIBDESI	I-O descriptor for index region	TABLE 9.3.3
53	184	IFIBBUFI	index region Buffer	-
237	208	-	a similar table for the overflow region	-
445	2	IFIBDRIX	Directory Index of DFH of keyfile	B BR
447	1	IFIBDUN	Disk Unit Number of Keyfile	BINARY
448	-	IFIBDFK	memory Disk File Header for the Key file	-
448	2	IFIBLA1	Length of first disk Area	B BR
450	2	IFIBDA1	sector address of first Disk Area	B BR
452	60	-	15 more length-address pairs	-
512	-	IFIBEND	-	LABEL

MAP IFIB (3.01) - INDEXED FILE INFORMATION BLOCK

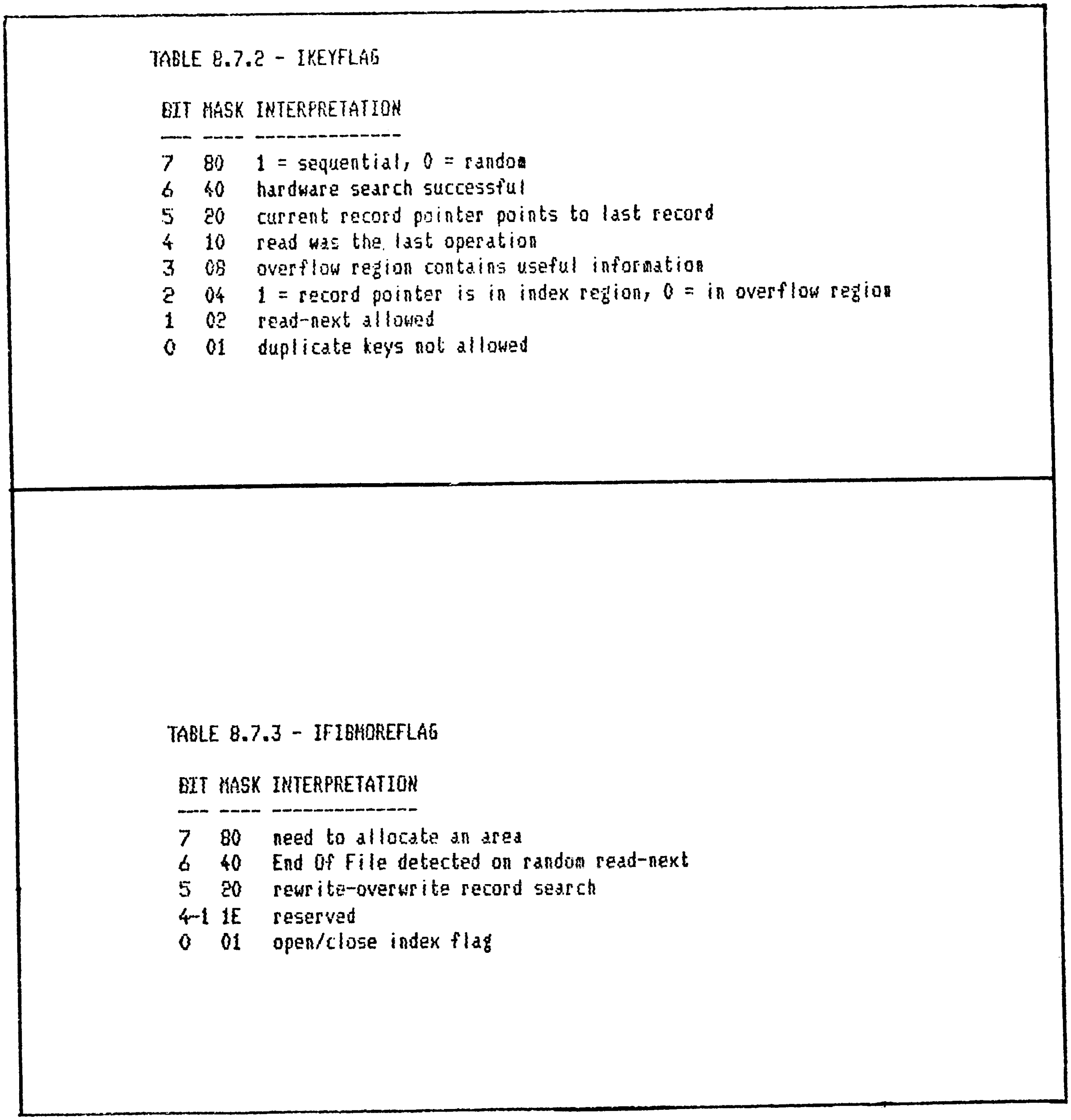


TABLE 8.7.2 - IKEYFLAG

BIT	MASK	INTERPRETATION
7	80	1 = sequential, 0 = random
6	40	hardware search successful
5	20	current record pointer points to last record
4	10	read was the last operation
3	08	overflow region contains useful information
2	04	1 = record pointer is in index region, 0 = in overflow region
1	02	read-next allowed
0	01	duplicate keys not allowed

TABLE 8.7.3 - IFIBMOREFLAG

BIT	MASK	INTERPRETATION
7	80	need to allocate an area
6	40	End Of File detected on random read-next
5	20	rewrite-overwrite record search
4-1	1E	reserved
0	01	open/close index flag

IKEYFLAG and
IFIBMOREFLAG

SECTION 9

9. LOGICAL AND PHYSICAL I-O

The task file structures (FPBs and FIBs) (see section 8) relate to the actual input-output media through Peripheral Handling Tables (PHTs) and queues of I-O descriptors. This section discusses how I-O operations are achieved by the MCP code concerned: Master Interrupt Poocessor (MIP), I-O Queue Handler, Device Dependent Routines (DDRs), Master Communicate Handler (MCH) and the Openclose slice.

The contents of the PHTs and I-O queues are often important for solving problems, especially where hardware faults are concerned. They can be printed using the PRINT PHT option of PMB80.

9.1 COMMUNICATES - MCH AND OPEN CLOSE

All input-output operations are performed by the MCP. A task issues a communicate which is decoded by the Master Communicate Handler. Not all communicates are I-O oriented, and I-O communicates do not always initiate a physical input-output operation.

Although the MCP performs its own I-O operations to devices, all task I-O is performed through a logical file structure. This structure is centered on a File Information Block. All logical I-O operations (open, close, read, write) are performed by a task through this FIB.

The MCP Openclose slice (slice 17), contains a Configuration Table (Table 9.1.1 - CT MAP) which contains logical device information including file-id, multifile-id, and logical device status.

The CT has an entry for each device on the system. When a file-open communicate is issued by a task, a link is established between the FIB and the CT entry. At the same time, a link is established between the FIB and the PHT (see Section 9.2) for the channel which accesses the device. The FIB is initialised with informaton from these tables and space is reserved for file buffers. These links, which are illustrated in Figure 9.1.1, should be checked on any dump where a failure in logical I-O is suspected.

The CT is maintained by the AVR task. The contents of the CT are analysed by the PRINT OL option of PMB80.

9.2 PERIPHERAL HANDLING - MIP AND DDRS

The actual work of input-output operations is performed by the Master Interrupt Processor and the Device Dependent Routines. The MIP performs only the simplest "read-more" or "write-more" controller commands. Ihe DDR for the particular controller handles the more complicated I-O operations, which are usually dependent on the controller involved.

Information on the physical status of the channel controller and device, and the logical status of the DDR for that device, are stored in a Peripheral Handling Table (Table 9.2.1 - PHTH MAP) and its device dependent extension. The field PHT.CIRC.BUF is a circular buffer of historical status information. Each entry is two bytes and the field PHT.CIRC.PTR is the index of the latest entry. The first byte of each entry is the device status received from the controller (see Tables 9.2.4 to 9.2.8): the second byte is the S-flags which are used by the MIP and DDRs to determine the activity which is currently being performed on the channel.

The interpretation of the device status byte is dependent on the appropriate hardware controller. S-flags consist of two parts. The most significant digit defines the type of operation being performed. The least significant digit is used, in combination with the channel status, to determine which MIP or DDR code should be performed. Each DDR contains a switch table to achieve this analysis; these details are not normally required, and are not discussed further in this document.

As a general rule, if the most significant bit of the device status is reset then the cause of the problem may be in this area, and should be investigated further.

9.3 INPUT-OUTPUT OPERATIONS - I-O QUEUE HANDLER

The PHTs are located at the high-address end of page zero memory. Each PHT holds one or more I-O descriptor queue heads (Table 9.3.1 - QHEAD MAP).

All I-O operations are performed as specified by the top I-O descriptor on the appropriate channel PHT queue. The actual I-O descriptor is found at a memory location which depends upon its origin (for example, the Virtual Memory I-O descriptor is located in the global MCP map VMWA, and descriptors for task-initiated I-O operations are located within the appropriate FIB in the locked TCB slice.

The descriptor queues are analysed by the PRINT PHT option of PMB80, and the queue structure should be checked for consistency, starting with the queue heads and following the links to the descriptors.

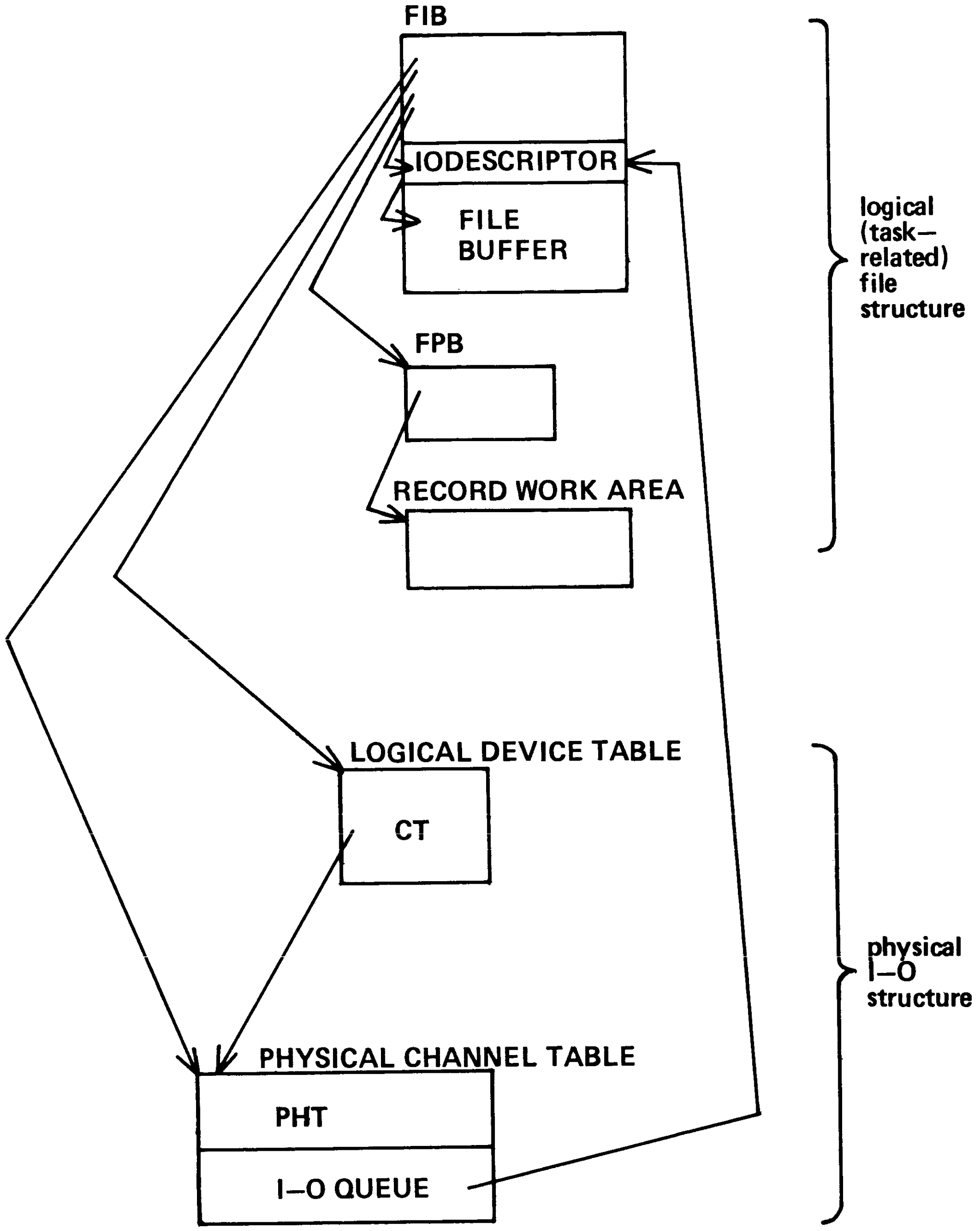
A queue head contains a flag field (Table 9.3.2) giving the status of the queue, and current descriptor and final descriptor memory addresses.

Note that the queue head contained within DISK PHTs does not point to a queue of I-O descriptors but to the disk parameter area (Table 9.2.3 - DSKPM MAP).

Each I-O descriptor has the same format (Table 9.3.3 - IODESC MAP). Console I-O however, uses some of the fields in a different manner

Field IODFL indicates the operation to be performed, and the result of the I-O operation when completed (see Table 9.3.5). Field IODPM is the task-id (Table 7.6.2) of the task which requested the operation.

9.1.1 LOGICAL AND PHYSICAL I-O STRUCTURE



LOGICAL AND PHYSICAL I-O STRUCTURE

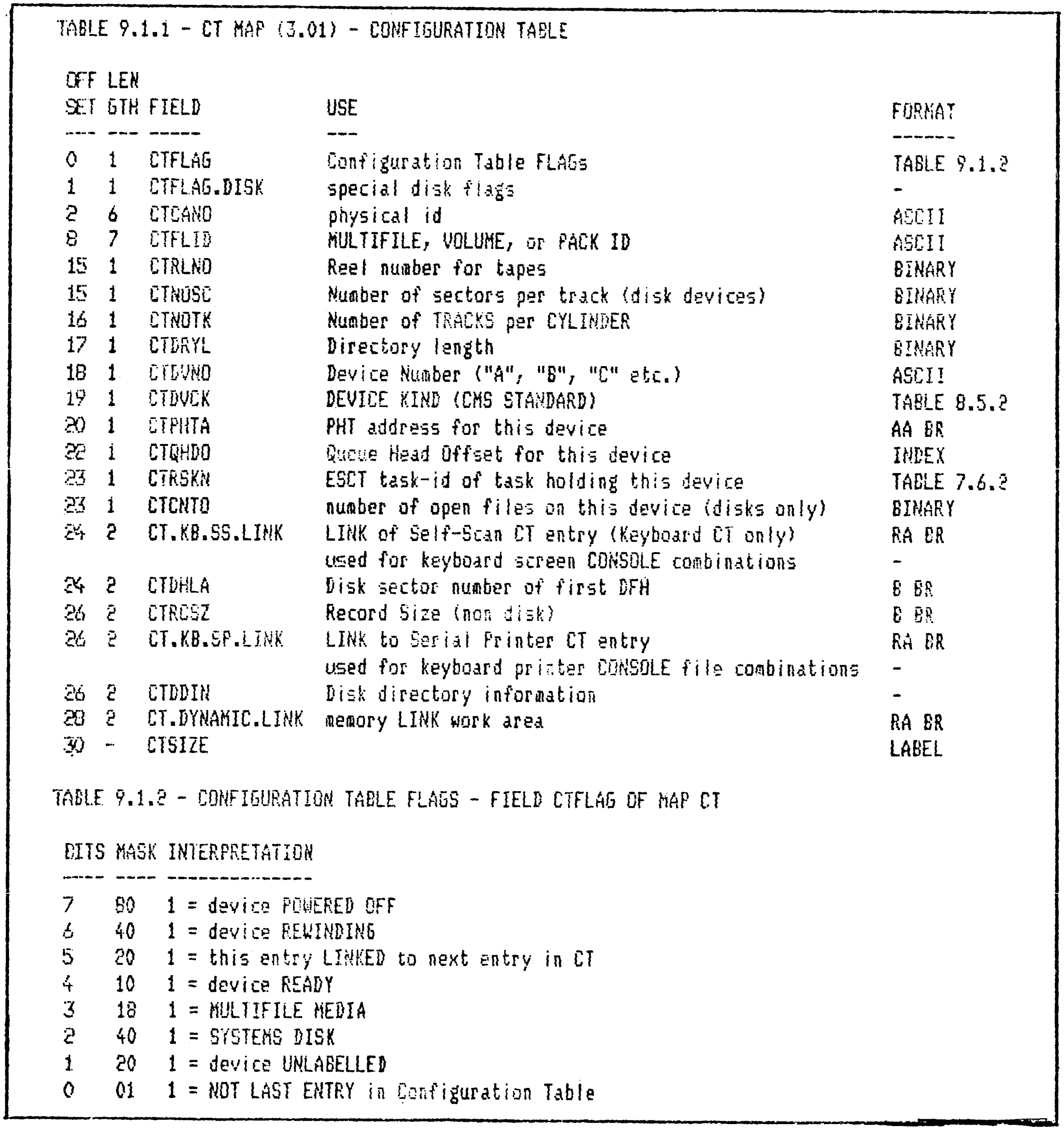


TABLE 9.1.1 - CT MAP (3.01) - CONFIGURATION TABLE

OFF SET	LEN GTH	FIELD	USE	FORMAT
0	1	CTFLAG	Configuration Table FLAGs	TABLE 9.1.2
1	1	CTFLAG.DISK	special disk flags	-
2	6	CTCANO	physical id	ASCII
8	7	CTFLID	MULTIFILE, VOLUME, or PACK ID	ASCII
15	1	CTRLNO	Reel number for tapes	BINARY
15	1	CTNOSC	Number of sectors per track (disk devices)	BINARY
16	1	CTNOTK	Number of TRACKS per CYLINDER	BINARY
17	1	CTDRYL	Directory length	BINARY
18	1	CTDVNO	Device Number ("A", "B", "C" etc.)	ASCII
19	1	CTDVCK	DEVICE KIND (CMS STANDARD)	TABLE 8.5.2
20	1	CTPHTA	PHT address for this device	AA BR
22	1	CTQHDO	Queue Head Offset for this device	INDEX
23	1	CTRSKN	ESCT task-id of task holding this device	TABLE 7.6.2
23	1	CTCNTO	number of open files on this device (disks only)	BINARY
24	2	CT.KB.SS.LINK	LINK of Self-Scan CT entry (Keyboard CT only)	RA BR
			used for keyboard screen CONSOLE combinations	-
24	2	CTDHLA	Disk sector number of first DFH	B BR
26	2	CTRCSZ	Record Size (non disk)	B BR
26	2	CT.KB.SP.LINK	LINK to Serial Printer CT entry	RA BR
			used for keyboard printer CONSOLE file combinations	-
26	2	CTDDIN	Disk directory information	-
28	2	CT.DYNAMIC.LINK	memory LINK work area	RA BR
30	-	CTSIZE		LABEL

TABLE 9.1.2 - CONFIGURATION TABLE FLAGS - FIELD CTFLAG OF MAP CT

BITS	MASK	INTERPRETATION
7	80	1 = device POWERED OFF
6	40	1 = device REWINDING
5	20	1 = this entry LINKED to next entry in CT
4	10	1 = device READY
3	18	1 = MULTIFILE MEDIA
2	40	1 = SYSTEMS DISK
1	20	1 = device UNLABELLED
0	01	1 = NOT LAST ENTRY in Configuration Table

CT MAP (3.01) - CONFIGURATION TABLE and
CONFIGURATION TABLE FLAGS - FIELD CRFLAG OF MAP CT

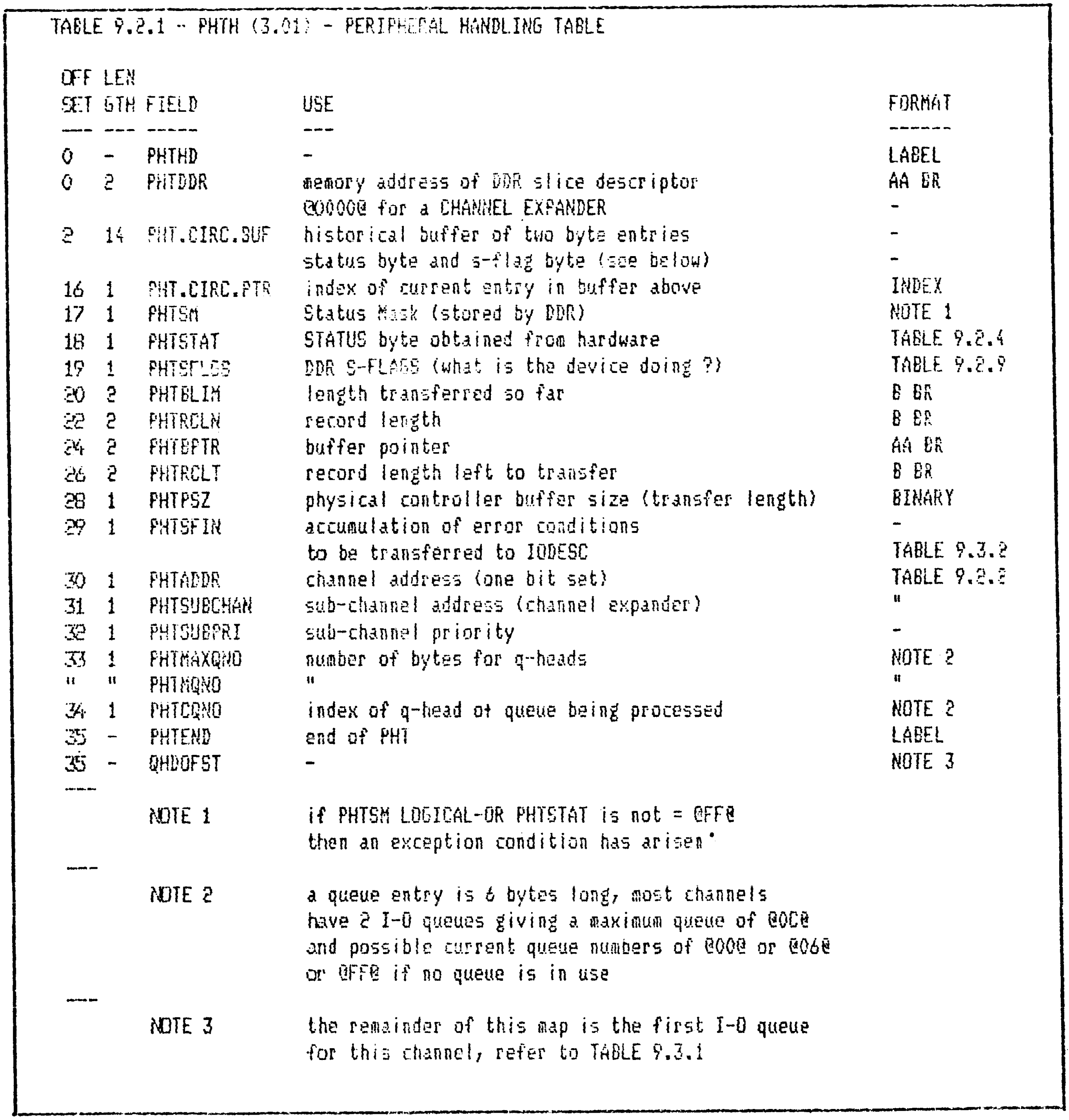


TABLE 9.2.1 - PHTH (3.01) - PERIPHERAL HANDLING TABLE

OFF SET	LEN GTH	FIELD	USE	FORMAT
---	---	-----	---	------
0	-	PHTHD	-	LABEL
0	2	PHTDDR	memory address of DDR slice descriptor	AA BR
			@0000@ for a CHANNEL EXPANDER	-
2	14	PHT.CIRC.BUF	historical buffer of two byte entries	-
			status byte and s-flag byte (see below)	-
16	1	PHT.CIRC.PTR	index of current entry in buffer above	INDEX
17	1	PHTSM	Status Mask (stored by DDR)	NOTE 1
18	1	PHTSTAT	STATUS byte obtained from hardware	TABLE 9.2.4
19	1	PHTSFLGS	DDR S-FLAGS (what is the device doing ?)	TABLE 9.2.9
20	2	PHTBLIM	length transferred so far	B BR
22	2	PHTRCLN	record length	B BR
24	2	PHTBPTR	buffer pointer	AA BR
26	2	PHTRCLT	record length left to transfer	B BR
28	1	PHTPSZ	physical controller buffer size (transfer length)	BINARY
29	1	PHTSFIN	accumulation of error conditions	-
			to be transferred to IODESC	TABLE 9.3.2
30	1	PHTADDR	channel address (one bit set)	TABLE 9.2.2
31	1	PHTSUBCHAN	sub-channel address (channel expander)	"
32	1	PHTSUBPRI	sub-channel priority	-
33	1	PHTMAXQNO	number of bytes for q-heads	NOTE 2
"	"	PHTMQNO	"	"
34	1	PHTCQNO	index of q-head ot queue being processed	NOTE 2
35	-	PHTEND	end of PHT	LABEL
35	-	QHDOFST	-	NOTE 3

NOTE 1 — if PHTSM LOGICAL-OR PHTSTAT is not = @FF@
then an exception condition has arisen˙

NOTE 2 — a queue entry is 6 bytes long, most channels
have 2 I-O queues giving a maximum queue of @0C@
and possible current queue numbers of @00@ or @06@
or @FF@ if no queue is in use

NOTE 3 — the remainder of this map is the first I-O queue
for this channel, refer to TABLE 9.3.1

PHTH (3.01) - PERIPHERAL HANDLING TABLE

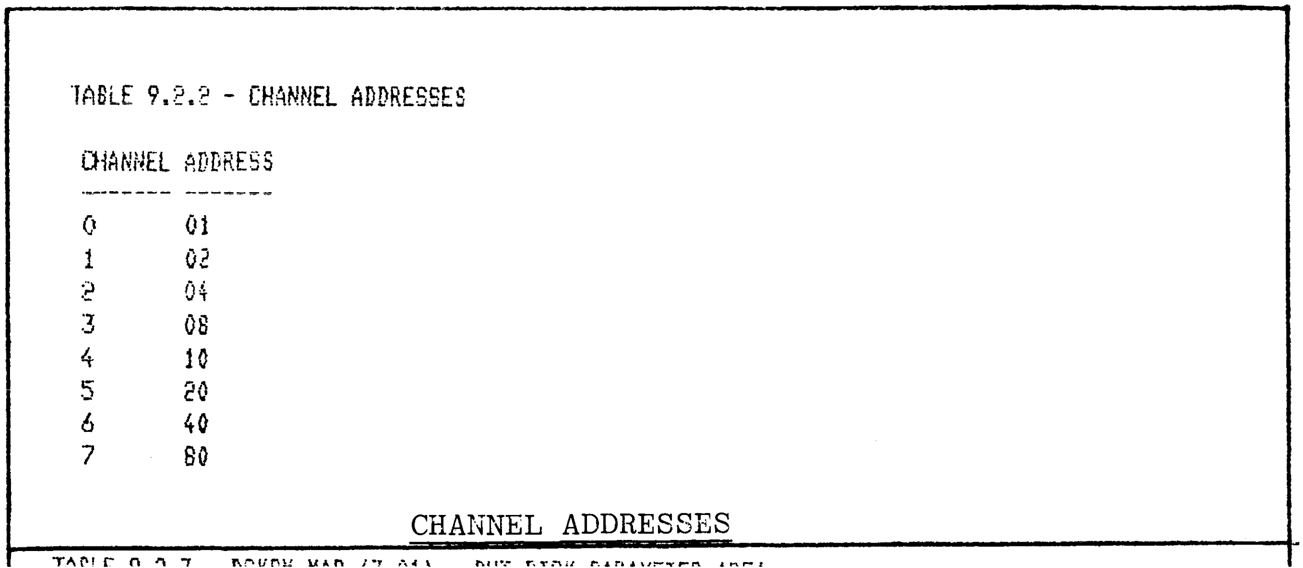


TABLE 9.2.2 - CHANNEL ADDRESSES

CHANNEL	ADDRESS
0	01
1	02
2	04
3	08
4	10
5	20
6	40
7	80

CHANNEL ADDRESSES

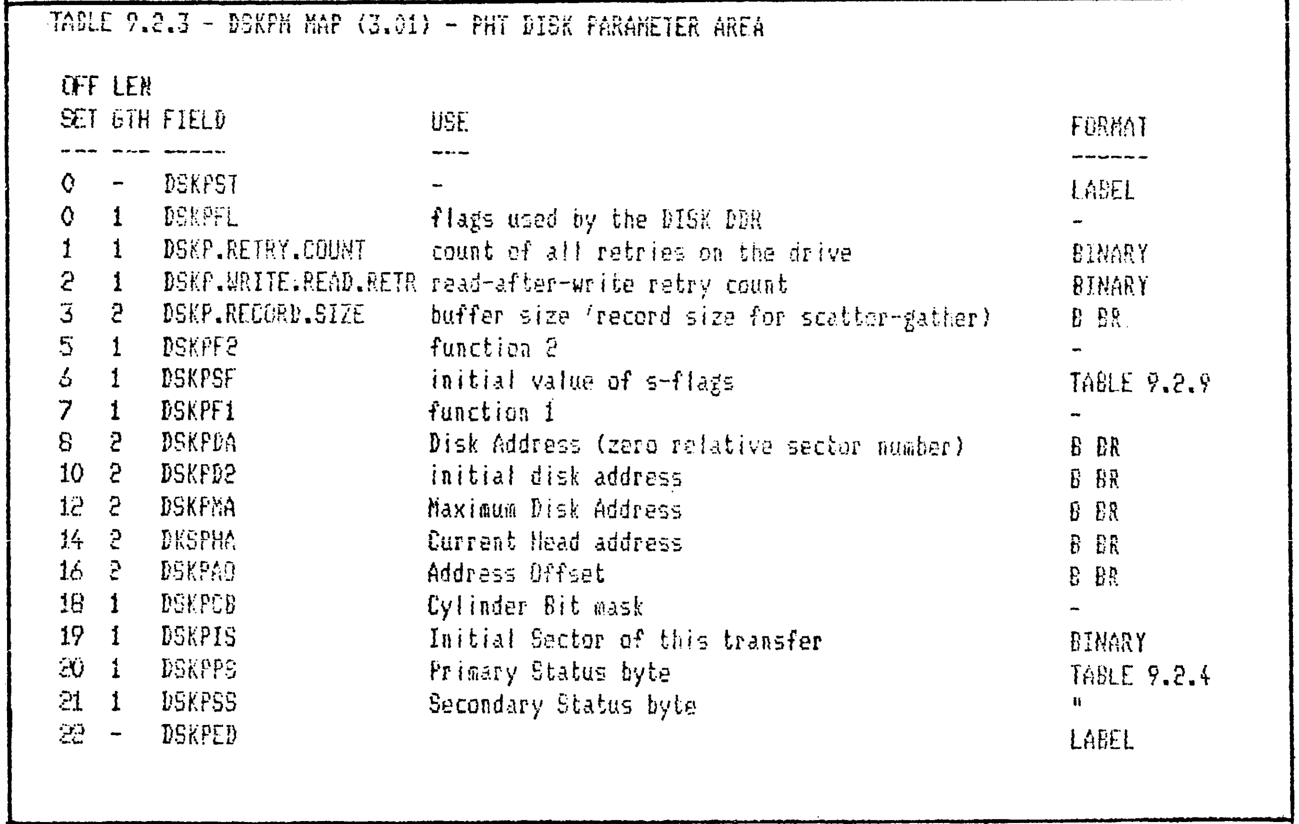


TABLE 9.2.3 - DSKPM MAP (3.01) - PHT DISK PARAMETER AREA

OFF SET	LEN GTH	FIELD	USE	FORMAT
0	-	DSKPST	-	LABEL
0	1	DSKPFL	flags used by the DISK DDR	-
1	1	DSKP.RETRY.COUNT	count of all retries on the drive	BINARY
2	1	DSKP.WRITE.READ.RETR	read-after-write retry count	BINARY
3	2	DSKP.RECORD.SIZE	buffer size (record size for scatter-gather)	B BR
5	1	DSKPF2	function 2	-
6	1	DSKPSF	initial value of s-flags	TABLE 9.2.9
7	1	DSKPF1	function 1	-
8	2	DSKPDA	Disk Address (zero relative sector number)	B BR
10	2	DSKPD2	initial disk address	B BR
12	2	DSKPMA	Maximum Disk Address	B BR
14	2	DKSPHA	Current Head address	B BR
16	2	DSKPAO	Address Offset	B BR
18	1	DSKPCB	Cylinder Bit mask	-
19	1	DSKPIS	Initial Sector of this transfer	BINARY
20	1	DSKPPS	Primary Status byte	TABLE 9.2.4
21	1	DSKPSS	Secondary Status byte	"
22	-	DSKPED		LABEL

DSKPM MAP (3.01) - PHT DISK PARAMETER AREA

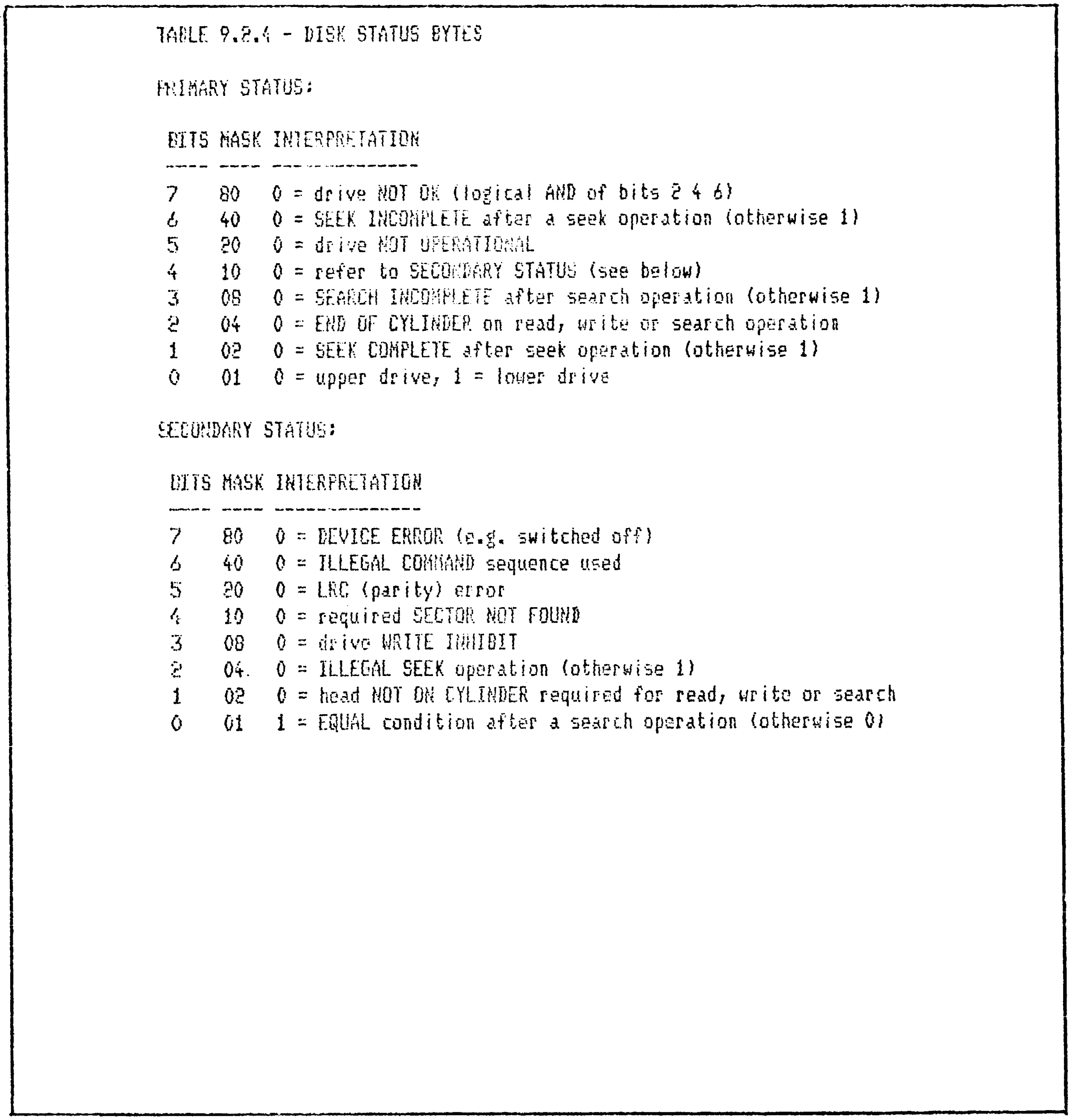


TABLE 9.2.4 - DISK STATUS BYTES

PRIMARY STATUS:

BITS	MASK	INTERPRETATION
7	80	0 = drive NOT OK (logical AND of bits 2 4 6)
6	40	0 = SEEK INCOMPLETE after a seek operation (otherwise 1)
5	20	0 = drive NOT OPERATIONAL
4	10	0 = refer to SECONDARY STATUS (see below)
3	08	0 = SEARCH INCOMPLETE after search operation (otherwise 1)
2	04	0 = END OF CYLINDER on read, write or search operation
1	02	0 = SEEK COMPLETE after seek operation (otherwise 1)
0	01	0 = upper drive, 1 = lower drive

SECONDARY STATUS:

BITS	MASK	INTERPRETATION
7	80	0 = DEVICE ERROR (e.g. switched off)
6	40	0 = ILLEGAL COMMAND sequence used
5	20	0 = LRC (parity) error
4	10	0 = required SECTOR NOT FOUND
3	08	0 = drive WRITE INHIBIT
2	04.	0 = ILLEGAL SEEK operation (otherwise 1)
1	02	0 = head NOT ON CYLINDER required for read, write or search
0	01	1 = EQUAL condition after a search operation (otherwise 0)

DISK STATUS BYTE

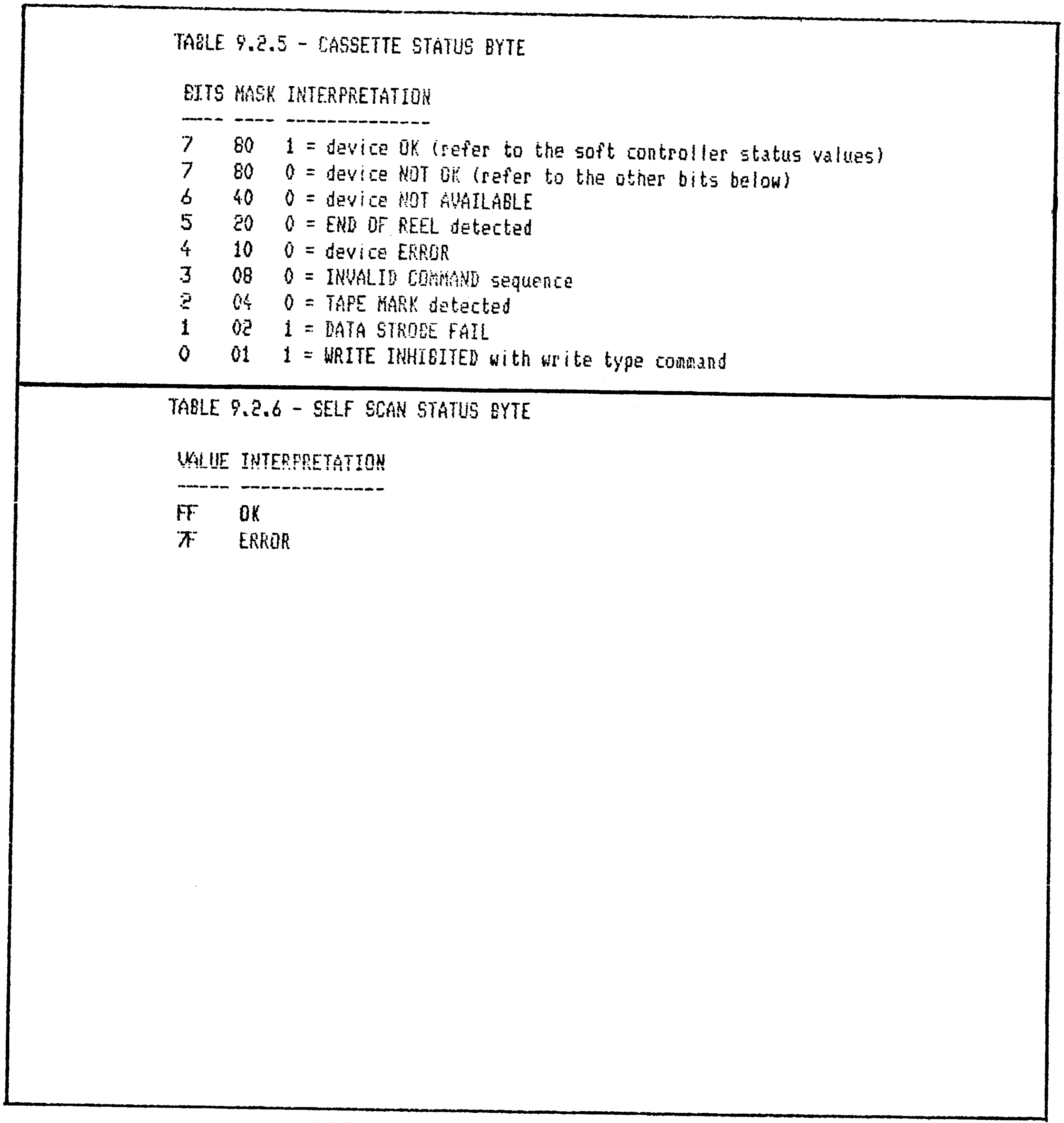


TABLE 9.2.5 - CASSETTE STATUS BYTE

BITS	MASK	INTERPRETATION
7	80	1 = device OK (refer to the soft controller status values)
7	80	0 = device NOT OK (refer to the other bits below)
6	40	0 = device NOT AVAILABLE
5	20	0 = END OF REEL detected
4	10	0 = device ERROR
3	08	0 = INVALID COMMAND sequence
2	04	0 = TAPE MARK detected
1	02	1 = DATA STROBE FAIL
0	01	1 = WRITE INHIBITED with write type command

TABLE 9.2.6 - SELF SCAN STATUS BYTE

VALUE	INTERPRETATION
FF	OK
7F	ERROR

CASSETTE STATUS BYTE and VALUE INTERPRETATION

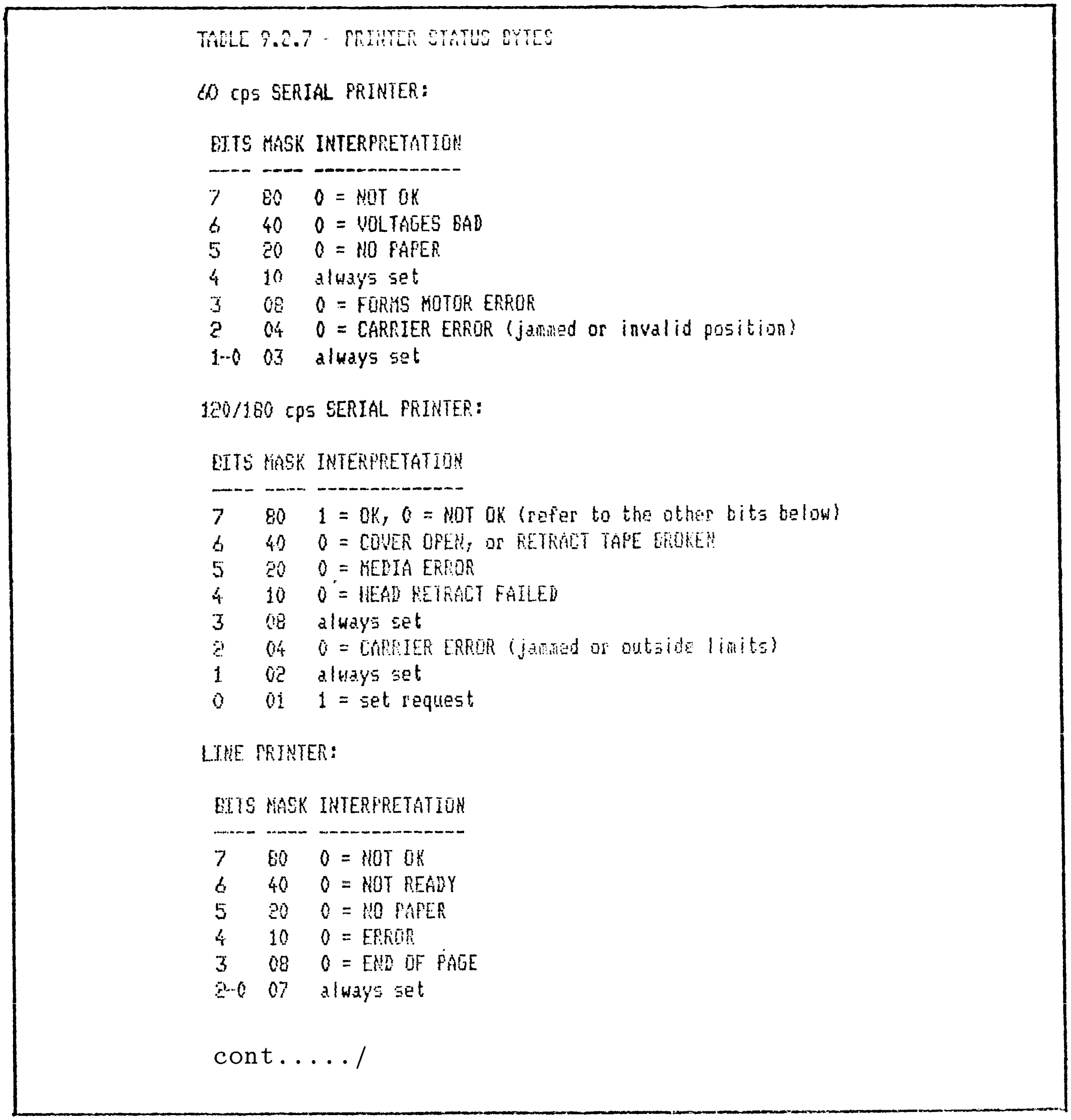


TABLE 9.2.7 - PRINTER STATUS BYTES

60 cps SERIAL PRINTER:

BITS	MASK	INTERPRETATION
7	80	0 = NOT OK
6	40	0 = VOLTAGES BAD
5	20	0 = NO PAPER
4	10	always set
3	08	0 = FORMS MOTOR ERROR
2	04	0 = CARRIER ERROR (jammed or invalid position)
1-0	03	always set

120/180 cps SERIAL PRINTER:

BITS	MASK	INTERPRETATION
7	80	1 = OK, 0 = NOT OK (refer to the other bits below)
6	40	0 = COVER OPEN, or RETRACT TAPE BROKEN
5	20	0 = MEDIA ERROR
4	10	0 = HEAD RETRACT FAILED
3	08	always set
2	04	0 = CARRIER ERROR (jammed or outside limits)
1	02	always set
0	01	1 = set request

LINE PRINTER:

BITS	MASK	INTERPRETATION
7	80	0 = NOT OK
6	40	0 = NOT READY
5	20	0 = NO PAPER
4	10	0 = ERROR
3	08	0 = END OF PAGE
2-0	07	always set

cont...../

PRINTER STATUS BYTES

TABLE 9.2.7 - PRINTER STATUS BYTES (cont.)

VALUE	INTERPRETATION
>7F	OK
7B	(60,120,180) CARRIER JAM
77	(60) FORMS JAM
6F	(120,180) HEAD RETRACT ERROR
5F	(60,120,180,LP) FORMS ERROR
3F	(60,120,180,LP) ERROR

TABLE 9.2.8 - KEYBOARD STATUS BYTE

VALUE	INTERPRETATION
FF	OK
6F	BUFFER OVERFLOW
3F	DISCONNECTED

PRINTER STATUS BYTES (cont.) and
KEYBOARD STATUS BYTE

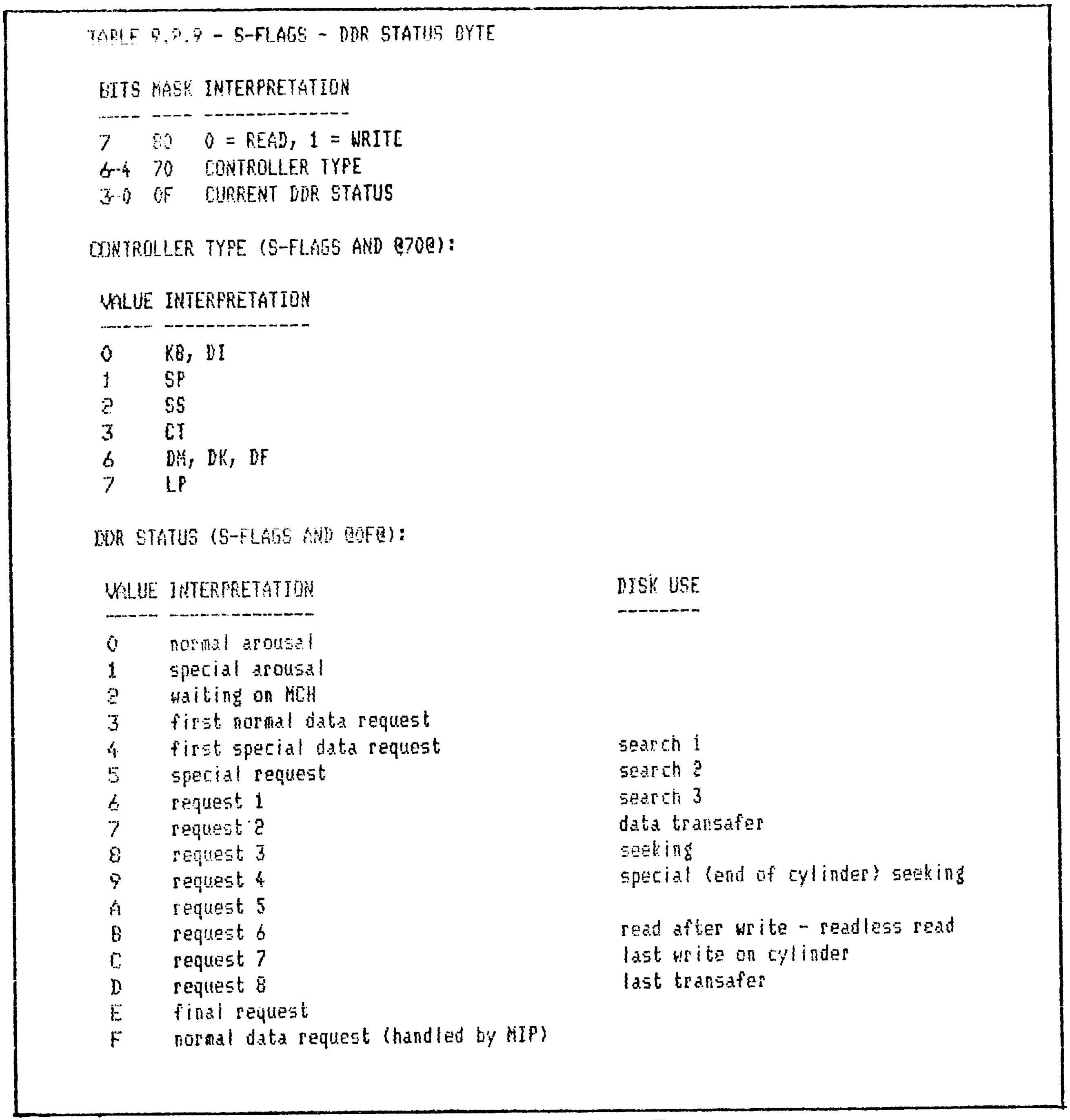


TABLE 9.2.9 - S-FLAGS - DDR STATUS BYTE

BITS	MASK	INTERPRETATION
7	80	0 = READ, 1 = WRITE
6-4	70	CONTROLLER TYPE
3-0	0F	CURRENT DDR STATUS

CONTROLLER TYPE (S-FLAGS AND @70@):

VALUE	INTERPRETATION
0	KB, DI
1	SP
2	SS
3	CT
6	DM, DK, DF
7	LP

DDR STATUS (S-FLAGS AND @0F@):

VALUE	INTERPRETATION	DISK USE
0	normal arousal	
1	special arousal	
2	waiting on MCH	
3	first normal data request	
4	first special data request	search 1
5	special request	search 2
6	request 1	search 3
7	request 2	data transafer
8	request 3	seeking
9	request 4	special (end of cylinder) seeking
A	request 5	
B	request 6	read after write - readless read
C	request 7	last write on cylinder
D	request 8	last transafer
E	final request	
F	normal data request (handled by MIP)	

S-FLAGS - DDR STATUS BYTE

TABLE 9.3.1 - QHEAD MAP (3.01) - IO DESCRIPTOR QUEUE HEAD

OFF	LEN			
SET	GTH	FIELD	USE	FORMAT
0	1	QFL	Queue FLags	TABLE 9.3.2
1	2	QCD	address of field IODLK of first IODESC (TABLE 9.3.3)	AA BR
3	2	QFD	address of final IO descriptor on queue	AA BR
5	1	QDA	Drive Address (multidrive controllers)	-
6	-	QHE	-	LABEL
6	-	QHDSZ	-	LABEL

TABLE 9.3.2 - QUEUE FLAGS - FIELD QFL OF MAP QHEAD

BITS	MASK	INTERPRETATION
7-6	C0	reserved
5	20	1 = queue empty
4	10	1 = queue not ready
3	08	1 = readiness changed
2	04	1 = return errors if not ready
1	02	1 = dequeue descriptors if not ready
0	01	1 = AVR help task must be run

QHEAD MAP (3.01) - IO DESCRIPTOR QUEUE HEAD and QUEUE FLAGS - FIELD QFL OF MAP QHEAD

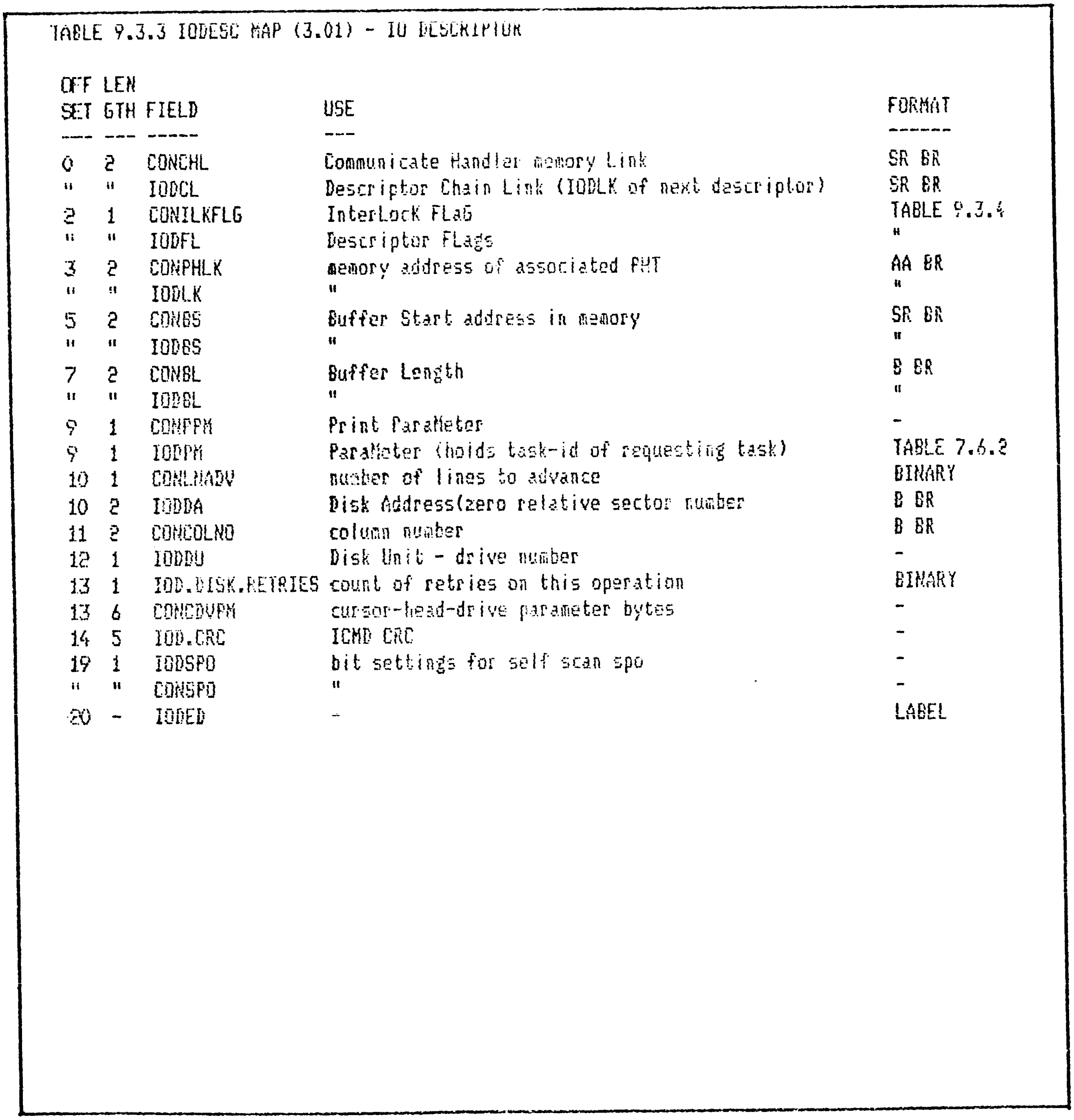
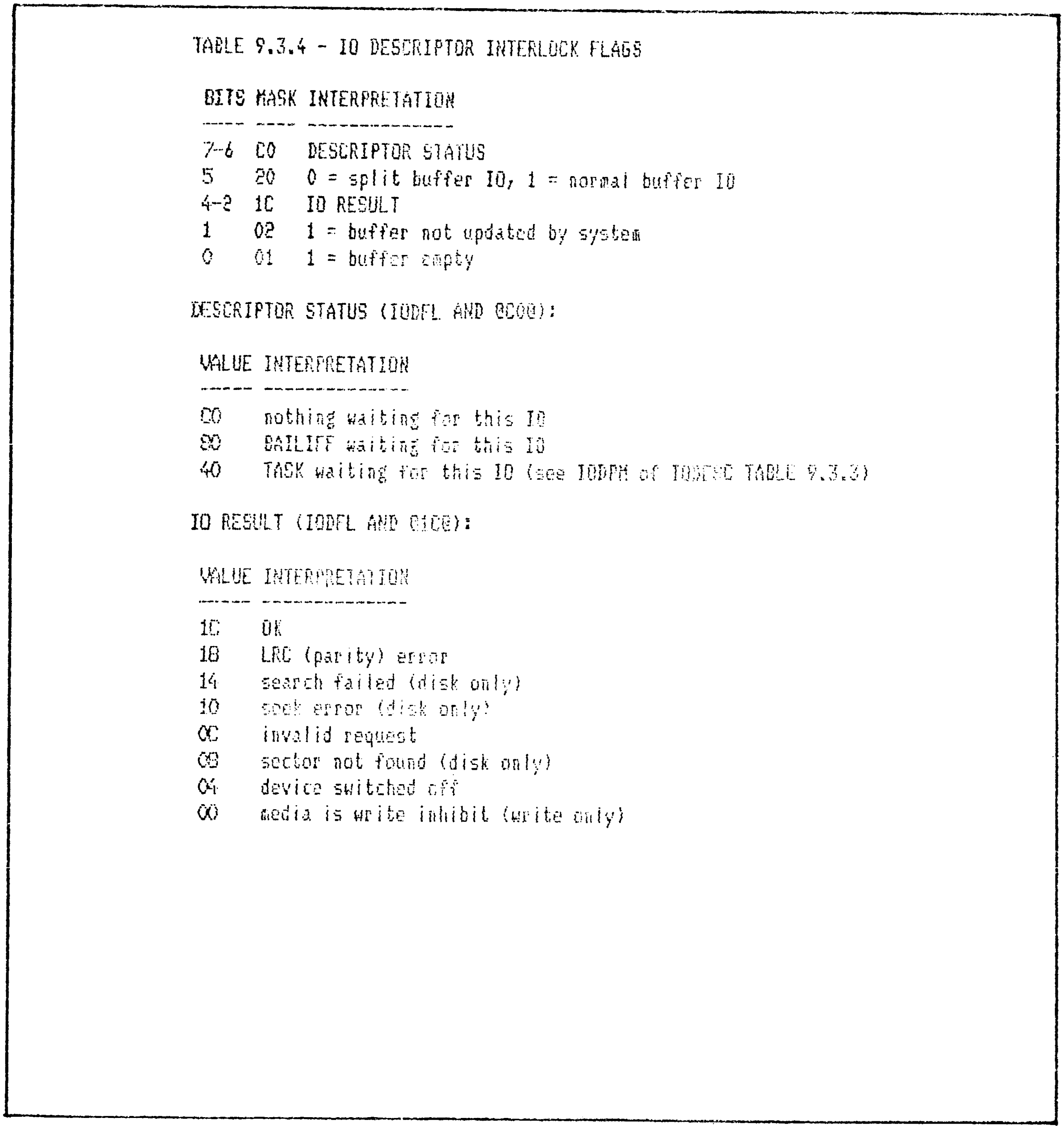


TABLE 9.3.3 IODESC MAP (3.01) - IO DESCRIPTOR

OFF	LEN			
SET	GTH	FIELD	USE	FORMAT
0	2	CONCHL	Communicate Handler memory Link	SR BR
"	"	IODCL	Descriptor Chain Link (IODLK of next descriptor)	SR BR
2	1	CONILKFLG	InterLock FLaG	TABLE 9.3.4
"	"	IODFL	Descriptor FLags	"
3	2	CONPHLK	memory address of associated PHT	AA BR
"	"	IODLK	"	"
5	2	CONBS	Buffer Start address in memory	SR BR
"	"	IODBS	"	"
7	2	CONBL	Buffer Length	B BR
"	"	IODBL	"	"
9	1	CONPPM	Print ParaMeter	-
9	1	IODPM	ParaMeter (holds task-id of requesting task)	TABLE 7.6.2
10	1	CONLNADV	number of lines to advance	BINARY
10	2	IODDA	Disk Address(zero relative sector number	B BR
11	2	CONCOLNO	column number	B BR
12	1	IODDU	Disk Unit - drive number	-
13	1	IOD.DISK.RETRIES	count of retries on this operation	BINARY
13	6	CONCDVPM	cursor-head-drive parameter bytes	-
14	5	IOD.CRC	ICMD CRC	-
19	1	IODSPO	bit settings for self scan spo	-
"	"	CONSPO	"	-
20	-	IODED	-	LABEL

IODESC MAP (3.01) - IO DESCRIPTOR



```
TABLE 9.3.4 - IO DESCRIPTOR INTERLOCK FLAGS

 BITS MASK INTERPRETATION
 ---- ---- --------------
 7-6  C0   DESCRIPTOR STATUS
 5    20   0 = split buffer IO, 1 = normal buffer IO
 4-2  1C   IO RESULT
 1    02   1 = buffer not updated by system
 0    01   1 = buffer empty

DESCRIPTOR STATUS (IODFL AND 00C0):

 VALUE INTERPRETATION
 ----- --------------
 C0    nothing waiting for this IO
 80    BAILIFF waiting for this IO
 40    TASK waiting for this IO (see IODPM of IODESC TABLE 9.3.3)

IO RESULT (IODFL AND 01C0):

 VALUE INTERPRETATION
 ----- --------------
 1C    OK
 18    LRC (parity) error
 14    search failed (disk only)
 10    seek error (disk only)
 0C    invalid request
 08    sector not found (disk only)
 04    device switched off
 00    media is write inhibit (write only)
```

IO DESCRIPTOR INTERLOCK FLAGS

SECTION 10

10. CMS DISK ORGANISATION

The information on all disks used on CMS systems (excluding Industry Compatible Mini Disk - ICMD) is stored in files under the same logical structure. The layout and contents of a disk are sometimes relevant to dump analysis, and so a description of the CMS disk organisation is included in this document.

The format of Key files, program files, and program dump files is described, as this is also relevant to the analysis of some memory dumps.

The following utilities may be used to examine the information on a disk:

- PD shows the contents of the disk directory namelist
- KA analyses the disk directory and available table
- LIST may be used to print any sector (e.g. LIST SYSMEM 100 1)
- DA prints and formats various items

Refer to the CMS Software Operational Guide (form number 2007258) for operating instructions for these utilities.

10.1 DISK AREAS

Figure 10.1.1 shows the structure of the main elements of CMS disk organisation, which are: Track zero, a non-file directory: a file directory consisting of a namelist and a disk file header list: and the remainder of the disk which is divided into data areas.

10.2 TRACK ZERO

Track zero of CMS disks is used for two basic functions: a disk label which identifies the disk; and a bootstrap of machine code to enable the CMS system software to be started using this disk.

10.2 TRACK ZERO (CONT.)

10.2.1 DISK LABEL

The disk label is located at sector zero of all CMS disks (see Table 10.2.1 for the format). The contents of many of the fields in this label, especially pointers to the directory items, must be valid if the integrity of the information stored on the disk is to be preserved.

10.2.2 BOOTSTRAP

The bootstrap area lies in sectors 2 to 25 on all CMS-initialised disks. However, the bootstrap is written in machine code, which varies with the hardware used. For example, the B80 bootstrap is different from the B800 bootstrap. Only disks initialised on a CMS B80 system are guaranteed to contain a valid B80 bootstrap. Also, the bootstrap may change from release to release to accommodate new features.

Since the bootstrap code performs the memory dump routine (PK4 or PK5), it is important that the correct bootstrap is present in this reserved area of the disk used to bootstrap the system (PK2) when a memory dump is to be taken.

The bootstrap is written to sectors 2 to 25 by the IN and RF functions of the Stand-Alone Utility, which takes the information from the disk file called CMSBOOT.

Disks used to take memory dumps from a system running a given level of MCP should be initialised using the same level of Stand-Alone Utility and CMSBOOT file.

10.2.3 BAD AREA LOG

This lies in sectors 30 and 31 of all CMS disks. It is not however, fully implemented on the B80 3.01 system. Refer to section 10.3.1 for more information on bad areas.

10.3 DISK DIRECTORY

Although the three items of the disk directory are addressed independently by the disk label, the disk directory is usually a contiguous area on disk. Usually (though not on fixed disks initialised on the B80) this area follows directly after track zero (i.e. sector 32 onwards).

10.3.1 AVAILABLE TABLE

The area is also known as the "non-file directory" because it addresses those parts of the disk which are not allocated to disk files.

This section of the directory references all areas of the disk which are not assigned (see Table 10.3.1). When the disk initialise routine discovers a bad area of disk, an entry is made in the available table with length zero and start and end addresses reversed. On some disks, areas are reserved for Field Engineering purposes; this is achieved by considering the areas to be "bad". A "ghost entry" is also made in the available table which has a start address equal to zero and an end address equal to the last sector address on the disk +1, and a length of zero.

10.3.2 FILE DIRECTORY - NAME LIST

This is a contiguous block of sectors containing the names of the files on the disk (see Table 10.3.2). The MCP may reserve areas of disk by inserting a temporary entry with #82 in each byte of the name field. Unused entries have #81 in each byte of the name field. Each entry references an entry in the other part of the disk file directory, the DFH list, in the same relative position.

10.3.3 DISK FILE HEADER (DFH) LIST

Each sector in this area may contain a Disk File Header. The disk file header includes the detailed information about the file which it references. This information includes the type of file, the number of tasks using the file, and the location of the file areas. (See Table 10.3.3).

Allocation and de-allocation of disk space consists of updating both the non-file directory and the file directory. If the system fails while this is in process, then areas of disk may not be assigned, or may become assigned twice. This symptom may be determined by executing the KA utility for the disk.

10.3 DISK DIRECTORY (CONT.)

10.3.4 SYSMEM FILE

The first file in the directory of all CMS disks is called SYSMEM. The DFH for this file references the entire disk from sector 0 to the end of the disk. This file may not however be accessed by user programs, and does not appear in the analysis performed by the system utilities such as PD.

10.4 KEY AND TAG FILES

CMS files with indexed organisation consist of two files on disk: the key or tag file, and the data file. The key file contains the record keys and relative record pointers to the data file. A tag file (or null key file) does not contain the record keys, and the indexed "file" may consequently only be accessed sequentially.

Figure 10.4.1 shows the layout of a key/tag file and Tables 10.4.1 and 10.4.2 describe the formats of the areas concerned.

The first sector of a key file is the Key File Parameter Block (KFPB). This is followed by a rough table, an index region, and an overflow region. The length of each entry in the key file is determined by the length of the key as follows:

key length not > 5 bytes	:	entry length = 8 bytes
key length not > 13 bytes	:	entry length = 16 bytes
key length not > 21 bytes	:	entry length = 24 bytes
key length not > 28 bytes	:	entry length = 32 bytes

10.5 PROGRAM (S-CODE) FILES

COBOL, RPG, MPLII and NDL program files all have the same basic structure (see figure 10.5.1). The file is one contiguous area on disk and has a Program Parameter Block (PPB) in the first sector (see table 10.5.1). The PPB contains reference information about the program and pointers to items within the code file.

The Program Segment Table (PST) (see Table 10.5.2) contains descriptors pointing to the S-code segments of the program. The data Segment Table (DST) (see table 10.5.2) contains descriptors referencing the data segments of the program. The CCB Preset Area (CCBPA) contains the COP Table for COBOL and RPG programs. The Internal File Name Block (IFNB) contains a list of file names of the files used by the program.

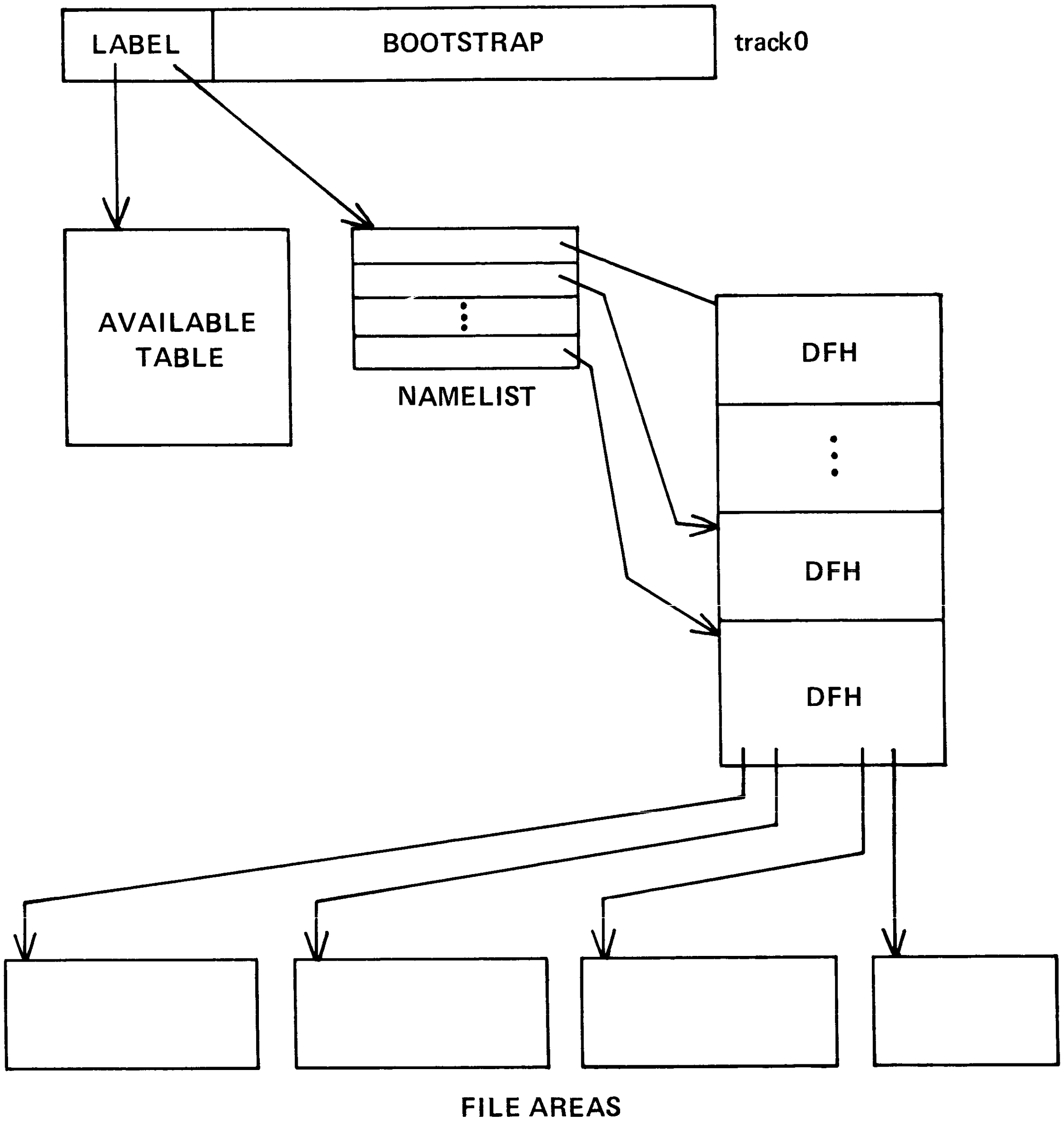
When a program is executed, these elements of the code file are used to build the Task Structure discussed in section 8.

10.6 PROGRAM DUMP FILES

A program dump file consists of three parts (see figure 10.6.1) which reside in two areas of disk. These three parts are the PPB, all the data segments (excluding FIBs), and the TCB. All of these items are obtained from the run structure of the task.

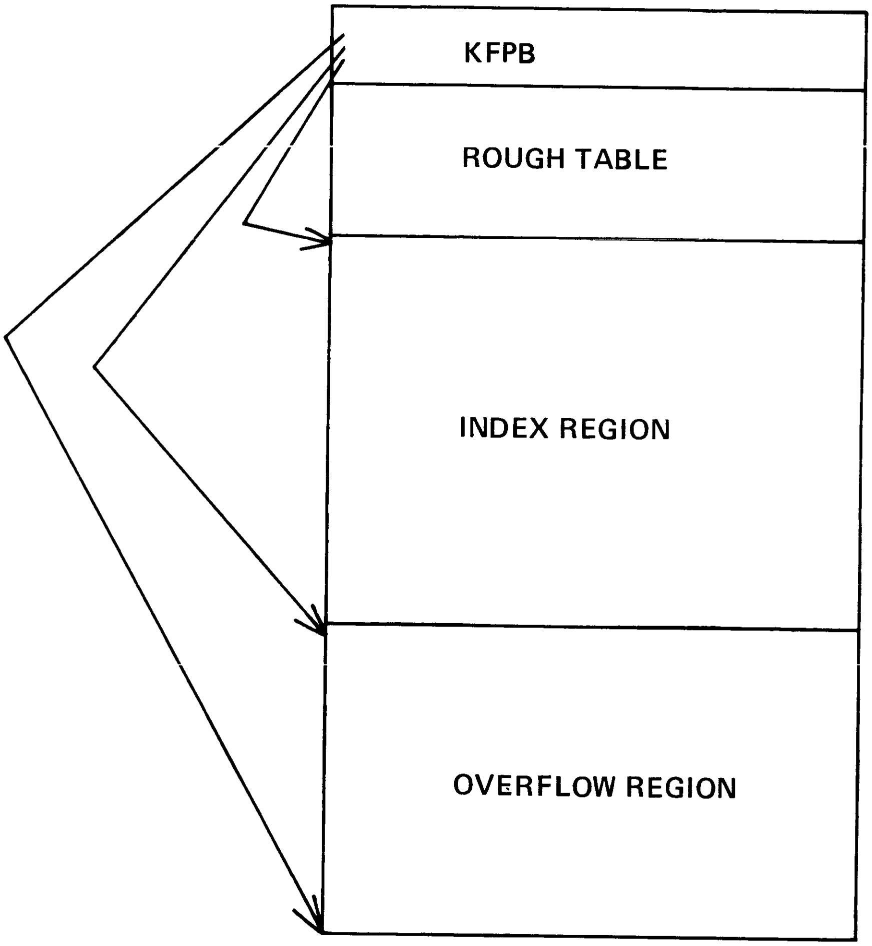
The PPB is obtained from the program code file with bytes 178-179 updated to contain the logical record number of the start of the TCB in the file. The data segments (excluding locked segments) are copied from the tasks's virtual memory file, all segments being updated from the memory copy, if the segment is present in memory as well as in the Virtual Memory file. Each segment starts on a sector boundary. The TCB is a copy of the TCB from memory, including all locked segments.

10.1.1 CMS DISK ORGANISATION



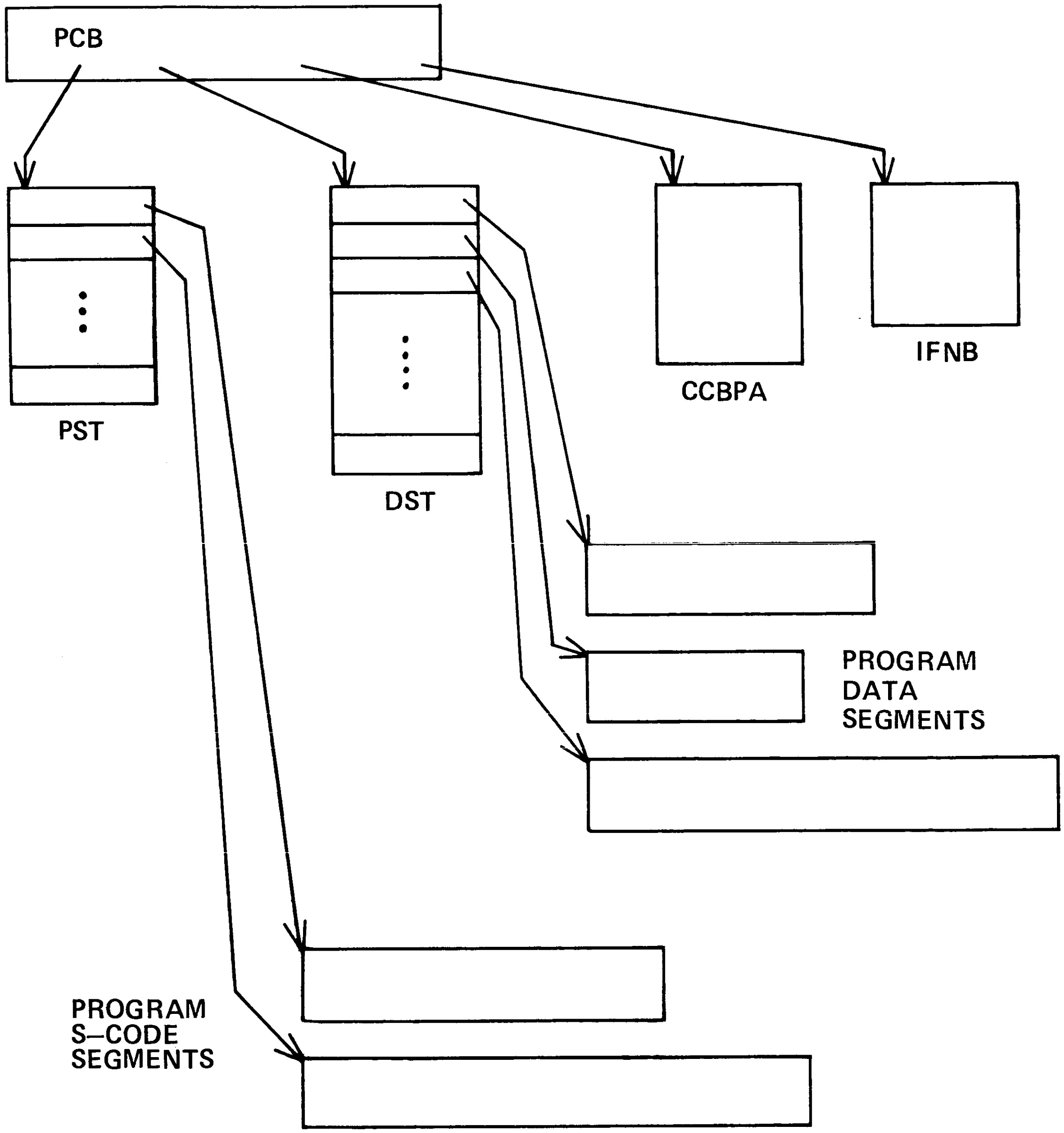
CMS DISK ORGANISATION

10.4.1 KEY FILE ORGANISATION



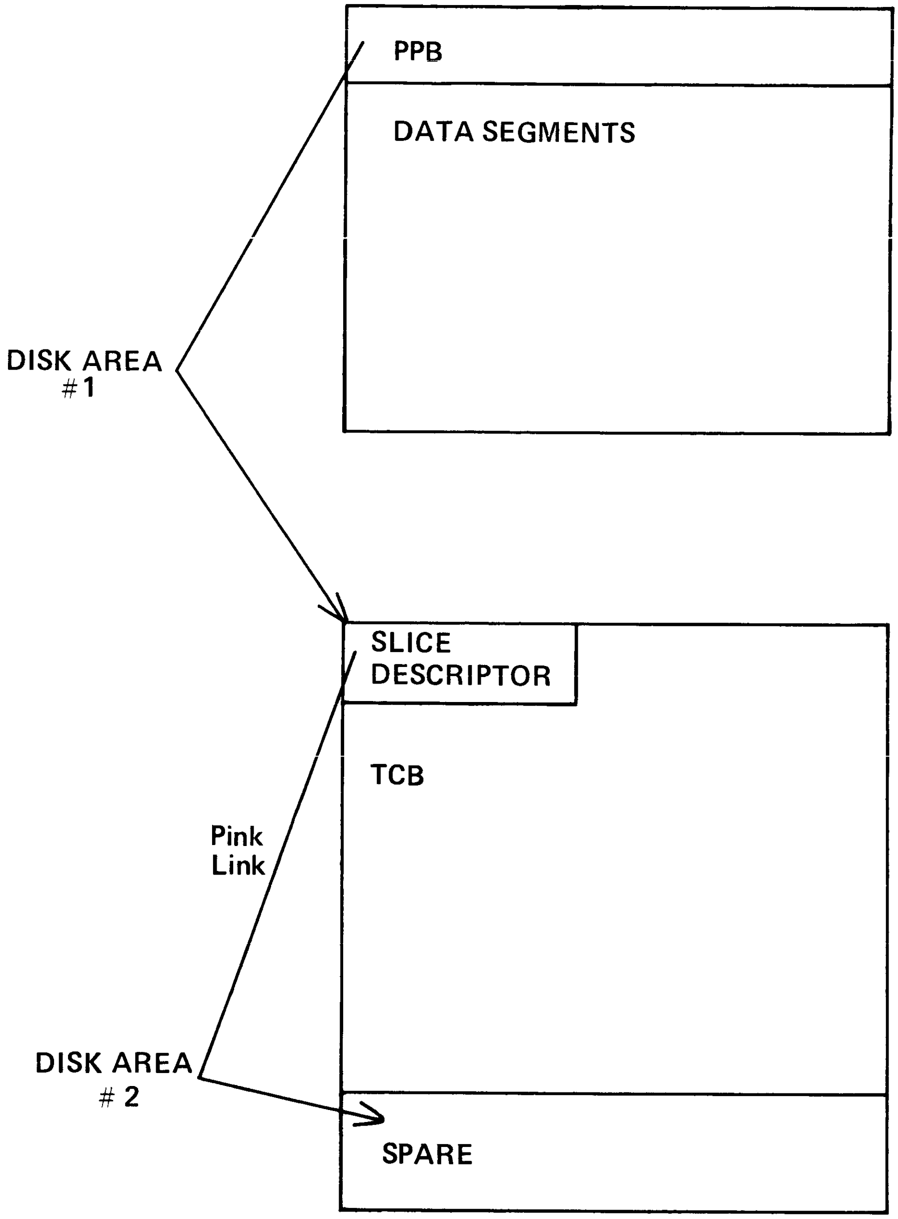
KEY FILE ORGANISATION

10.5.1 PROGRAM FILE ORGANISATION



PROGRAM FILE ORGANISATION

10.6.1 PROGRAM DUMP FILE STRUCTURE



PROGRAM DUMP FILE STRUCTURE

TABLE 10.2.1 - DISK LABEL

BYTE	CONTENTS	SIZE	DATA CODE
0-3	"VOL1"	04	E(BCDIC)
4-9	Serial no.	06	E
10	Blank (Access code)	01	E
11-20	"SL9INTERNL"	10	E
21-27	Cartridge identifier	07	A(SCII)
28-29	"S9" (System Interchange code)	02	E
30	Zero (Pack Code)	01	E
31-36	Reserved scratch	06	--
37-50	Owner's Identification	14	A
51-78	Reserved scratch	28	--
79	Blank	01	E
80-83	"VOL2"	04	E
84-88	Initialisation Date (YYDDD)	05	E
89-94	Initialising System (eg."BDS")	06	E
95	"R" if restricted cartridge	01	A
96-97	No. of Cylinders	02	B(INARY)
98	No. of Tracks-Cylinder	01	B
99	No. of Sectors-Track	01	B
100	No. of Sectors for Directory Name List	01	B
101-103	Sector Address of Directory Name List	03	B
104	No. of Sectors for Available Table	01	B

DISK LABEL

TABLE 10.2.1 - DISK LABEL cont.

BYTE	CONTENTS	SIZE	DATA CODE
105-107	Sector Adcress of Available Table	03	B
108-109	Maximum no. of files	02	B
110	Unit of Allocation(sectors)	01	B
111-113	Sector Address of First File Header	03	B
114-118	Reserved protected	05	--
119	Integrity Flag (0 = OK)	01	E
120-125	Actual Error Count	06	E
126-131	Bad Sector Count	06	E
132-135	Reserved for MTR	04	--
136-179	Reserved Scratch	34	--

DISK LABEL cont.

TABLE 10.2.2 - BAD AREA LOG

BYTES	CONTENTS
0-1	Total of bad allocation units recorded in this log
2-31	Binary zero. Reserved for possible future expansion
32-33	Address of first of group of contiguous bad allocation units
34-35	Number of allocation units in group
36-359	81 more address/length pairs

BAD AREA LOG

TABLE 10.3.1 - AVAILABLE TABLE BLOCK

BYTE	CONTENTS	SIZE	DATA CODE
0-1	Length of available area in allocation units **	02	B(INARY)
2-3	Address of first available allocation unit **	02	B
4-5	Address of last available allocation unit +1	02	B
6-173	28 more 6 byte entries	168	B
174-179	Sterile area (always zeroes)	06	B

** An unused entry has length = zero AND 1st and last allocation unit adcresses = 1.

AVAILABLE TABLE BLOCK

TABLE 10.3.2 - NAME LIST BLOCK

BYTE	CONTENTS	SIZE	DATA CODE
0-11	File Identifier	12	A(SCII)
12	Reserved (Blank character)	01	A
13	Directory Index	01	B(INARY)
14-15	Header Sector Address	02	B
16-175	Ten more 16 byte entries	160	A or B
176-179	Zero	04	B

Notes

Available directory entries. i.e. entries which do not at present correspond to files but could do so in the future contain #8C in each of the first 12 bytes of the entry. Bytes 14-15 of such entry contain the disk address of the sector holding this directory entry.

A directory entry whose corresponding file header is that of a temporary file, i.e. has not been closed with lock, contains #81 in each of the first 12 bytes of the entry and the sector address of the header in bytes 14-15.

Since each block of the directory name list can hold eleven entries and the total no. of headers may not be a multiple of eleven, it is possible that the last sector of the name list contains unusable entries. Such entries are marked by #82 in each of the first 12 bytes of the entry and zero in bytes 14-15.

The directory index gives the ordinal position of this entry within the directory. The number is recorced moculo 256.

NAME LIST BLOCK

TABLE 10.3.3 - DISK FILE HEADER (DFH)

BYTE	CONTENTS	SIZE	NOTES
0-11	File Identifier	12	AI
12	Blank	01	A
13	File Type(see table 6.4.4.2)	01	B
14-17	Flags	04	BF
18-22	Creation Date (YYDDD)	05	A
23-27	Last Access Date (YYDDD)	05	A
28-29	Record size (in bytes)	02	B
30-31	No. of records per block	02	B
32-33	No. of sectors per block	02	B
34	Implementation Level No.	01	B
35-37	Maximum file size (records)	03	B
38-40	Save factor (0-999)	03	A
41	Maximum area in use (0 = None)	01	B
42-43	No. of records in last area	02	B
44-45	Generation No.	02	B
46-47	No. of spare chars. in last record (Stream I-O)	02	B
48-54	Pack-id of overflow pack	07	A
55	User count	01	BU
56-57	Area Bit Map 1	02	BM
58-59	Area Bit Map 2	02	BM
60-61	Address of 1st File Area	02	BS
62-63	Size of 1st File Area	02	BS

DISK FILE HEADER (DFH)

TABLE 10.3.3 - DISK FILE HEADER cont.

BYTE	CONTENTS	SIZE	NOTES
64-123	15 more address-size pairs	60	B
124-128	Reserved for implementation dependant overflow pack pointers	05	B
129-131	No. of records in file	03	B

NOTES

A: ASCII characters

B: Binary number

F: Flags.

Bit 0 set = file has been "crunched"

Bit 1 set = rough table valid

Bit 2 set = file has section on overflow pack

Bit 3 set = single area file

Bit 4-31 are currently unassigned

I: File identifier

The contents of the file-id field are always the same as the contents of the file-id in the corresponding entry in the directory name list (even to being filled with #80 or #81for available entry and temporary entry respectively).

U : User counts

Bits 0-2 - total number of users (7=lccked)

Bit 3 - spare

Bit 4 - number of output users.

Bits 5-7 - number of lock access users

DISK FILE HEADER cont.

TABLE 10.3.3 - DISK FILE HEADER cont.

M : Area bit maps.

Area bit maps are 16 bit fields in which eact bit represents one of the 16 possible file areas. The most significant bit in the bit map corresponcs to the first file area and the least significant bit to the 16th area.

The bits in area bit map 1 have the following significance:-

Set = area allocated and on this pack.

Reset = are not allocated
or on other (overflow) pack.

The bits in area bit map 2 have the following significance.

set = area allocated and on other pack.
reset = area not allocated or on this pack.

S :-

Addresses anc sizes of file areas are in terms of the allocation unit of this pack which is an integer multiple of sectors and fixec at initialisation time.

Addresses for areas on an overflow pack are not necessarily correct. Sizes for areas on an overflow pack are correct and are given in terms of the allocation unit of this pack.

DISK FILE HEADER cont.

TABLE 10.3.4 - DFH FILETYPES

TYPE	CODE
Normal data ("D")	#00
Source language	#01 - #0E
Source library	#0F
Ordinary program (S-code)	#10 - #13
Interpreter for BDS	#14 - #17
Interpreter for B700	#18 - #1B
Interpreter for B170C	#1C - #1F
Sysmem	#20
VM file	#30
Indexed ("I")	#80 **
Key file ("K")	#81

** Value #8C never appears in file header but is used in FPB to indicate that an inoexed file is being opened.

FILETYPES

TABLE 10.4.1 - KEY FILE PARAMETER BLOCK (KFPB)

BYTE	CONTENTS	SIZE	NOTES
0	Implementation Level No.	01	B
1-2	Spare	02	-
3-9	Pack-id of data file	07	A
10-21	File-id of data file	12	A
22	Blank	01	A
23-27	Space for implementation cefined link to data file	05	-
28	KFPB flags - true if bit set Bit 0 : B80 created rough table Bit 1 : B700 created rough table Bit 2 : B1700 created rough table Bit 5 : Data file is a dual pack file Bit 7 : Duplicates allowed	01	B
29-31	Relative record no. at start of rough table	03	B
32-33	Length of rough table in sectors	02	B
34	Spare	01	-
35-37	Relative record no. of start of overflow region	03	B
38-40	Size of overflow region in sectors	03	B
41-43	Relative record no. of start of index region	03	B
44-46	Size of index region in sectors	03	B

KEY FILE PARAMETER BLOCK (KFPB)

TABLE 10.4.1 - KEY FILE PARAMETER BLOCK cont.

BYTE	CONTENTS	SIZE	NOTES
47	Spare	01	-
48-49	Size of key part in bytes	02	B
50-51	Offset of keypart from base of data record in bytes	02	B
52-55	Zero	04	BD
56-179	Spare	124	

NOTES:

A :	ASCII characters
B :	Binary number
C :	Set-Reset by close

KEY FILE PARAMETER BLOCK cont.

TABLE 10.4.2 - ROUGH TABLE ENTRY

BYTE 0 (ENTRYSZ - 4)	-	Highest key value in this group of index sectors,left justified,binary zero filled.
(ENTRYSZ - 3) (ENTRYSZ - 1)	-	Lowest sector address in this group of sectors.

NB
--

1) The format of rough table is implementation dependant. All implementations are, however, required to assign sufficient disk space to accomodate a rough table, formatted as above, with a group size of 32 sectors.

2) The rough table is not updated if the record corresponding to the highest key in a group of sectors is deleted.

ROUGH TABLE ENTRY

TABLE 10.4.3 - KEY ENTRY (INDEX OR OVERFLOW)

```
BYTE 0            -   Key value,left justified,binary zero filled.
(ENTRYSZ - 4)

(ENTRYSZ - 3)     -   Relative record no. within data file of
(ENTRYSZ - 1)         keyed record.

             NB
             --

               Deletion of a record  is  indicated  by  zeroing  the  key
               field.
```

KEY ENTRY (INDEX OR OVERFLOW)

TABLE 10.5.1 - PROGRAM PARAMETER BLOCK (PPB)

BYTES	PURPOSE OF FIELD	SIZE	REF	COMMENTS
0	Implementation level No,	1	B	
1-12	Program name.	12	A	Standarc 12 character file as in FPB.
13-24	S-language name.	12	A	For documentation.
25-31	Interpreter pack-ic.	7	A	
32-43	Interpreter name.	12	A	
44-55	Compiler name.	12	A	For documentation.
56-61	Compilation date.	6	A	YYMMDD.
62-63	Priority class.	2	B	See Table 4.2.1.2.
64	Data segment for initiating message.	1	B	#FF implies discard message.
65-67	S-program start address.	3	B	Segment\Displacement.
68-69	Program segment table length.	2	B	NO. of segments *6
70-71	P.S.T. location.	2	B	Logical record no. within this file.
72-73	Data segment table	2	B	NO. of segments *6.
74-75	C.S.T. location.	2	B	Logical record no. within this file.
76-77	TCB preset area length.	2	B	In bytes.
78-79	TCB preset area address.	2	B	Byte displacement within PPB.
80-81	(Partial)Stack length.	2	B	In bytes.
82-83	CCB preset area length.	2	B	In bytes.
84-85	CCB preset area address.	2	B	Logical record no. within this file.

PROGRAM PARAMETER BLOCK (PPB)

TABLE 10.5.1 - PROGRAM PARAMETER BLOCK (PPB) cont.

BYTES	PURPOSE OF FIELD	SIZE	REF	COMMENTS
86-87	TCB preset extension length.	2	B	In bytes.
88-89	Internal file name block length.	2	B	In bytes.
90-91	Internal file name block address.	2	B	Logical record within this file.
---	TCB preset area values.	----	---	Variable length.

PROGRAM PARAMETER BLOCK (PPB) cont.

TABLE 10.5.2 - PROGRAM SEGMENT DESCRIPTOR

Byte 0 TYPE CODE

value = 0 : ordinary code or data segment (bytes 2,3 = 0 implies a zero filled work area)

value = 1 : this read-write data segment is an FIB

value = 2 : a dummy entry will be built in the segment table (bytes 2,3,4,5 = 0)

value = 3 : uninitialised (garbage filled) work segment (bytes 2,3 = 0)

Byte 1 FLAGS

bit 0,1,2,3,4,5 used by OS

bit 6 set = lock in main store

bit 7 set = read-write segment (never set for code, must be set if bytes 2,3 = 0)

Bytes 2,3 Relative record number within file at start of segment

Bytes 4,5 Length of segment in bytes +

+ For an FIB segment descriptor, byte 5 contains the segment number of the appropriate FPB segment.

PROGRAM SEGMENT DESCRIPTOR

TABLE 10.5.3 - PROGRAM INTERNAL FILE NAME BLOCK ENTRY

Byte 0	DST index of FIB
Byte 1	DST Index of FPB
Bytes 2-29	Internal file name. Left justified and blank filled

PROGRAM INTERNAL FILE NAME BLOCK ENTRY

APPENDIX A

APPENDIX A - INDEX OF TERMS

APPENDIX A - INDEX OF TERMS (CONT.)