TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES

ABOUT THIS DOCUMENT

PURPOSE

The V Series MCP/VS System Software Support Reference Manual provides reference material about the MCP/VS operating system, and describes several tools used to support the operating system.

SCOPE

The reference material in this document is a limited subset of the information needed to support the MCP/VS operating system. The material included is the most useful material that can be formatted for brevity, quick reference and easy access.

This document does **not** contain a complete or detailed explanation of the operating system. It does not contain any training or tutorial sections for inexperienced users.

All of the material in this document is relative to the 2.0 release of the MCP/VS operating system. Support information for the 1.0 release of MCP/VS is not included.

Section 6 and appendix A contain information directly related to the hardware of a V 300 system. Sections 4, 7 and 9 contain smaller amounts of hardware-dependent information. The introduction of V Series systems other than the V 300 will definitely affect section 6 and appendix A, and may affect sections 4, 7 and 9.

AUDIENCE

This document is intended for the use of Unisys personnel who support the MCP/VS operating system. This document is not intended for customers and is not authorized for distribution to customers.

PREREQUISITES

Users of this document should be experienced in operating systems support and familiar with Unisys B 2000/B 3000/B 4000 Series or V Series systems. A familiarity with the MCPIX operating system (predecessor to MCP/VS) is assumed in some portions of the document.

Users of this document should become familiar with the other documents needed to understand and use the MCP/VS operating system. These documents are listed under Related Product Information later in this section.

HOW TO USE THIS DOCUMENT

This document is organized to be referenced quickly and easily. All items in the text are clearly marked with headings (which are duplicated in the table of contents).

- For information about support tools used with the MCP/VS operating system, see sections 6, 7, 8 and 9.
- For a description of the LINKER listing and the operating system information it contains, see section 2.

- For a description of MCP/VS memory dumps, see section 3. Explanations of some items contained in the memory dumps are also included.
- For information about the modules that make up the operating system (their names, functions and so forth), see section 1.
- For a description of I/O operations under MCP/VS with an emphasis on logical I/O, see section 5.
- For information about BCTs, pre-term codes, and soft result descriptors, see appendix A.
- For hardware-related information, see appendix A or refer to the hardware manuals referenced in appendix A. Section 6 also contains hardware-related information. A summary of the hardware instructions that change the state of the system is included in section 4.

ORGANIZATION

This document is organized as follows:

SECTION 1: OPERATING SYSTEM MODULES

This section lists the modules of the operating system, describes their basic functions and the major data structures that they operate on. Information about module names and a system command/module cross reference is also included.

SECTION 2: THE LINKER LISTING

This section describes the LINKER listing that is provided along with the source code listing of the operating system. This listing contains information about the way that the modules of the operating system are linked together into a code file.

SECTION 3: SYSTEM MEMORY DUMPS

This section provides information about when system memory dumps are produced (under what circumstances) and the command used to print system memory dumps. The section also includes descriptions of the formatted dump listings for several important operating system data structures.

SECTION 4: SUMMARIES OF STATE-CHANGING INSTRUCTIONS

This section contains conceptual summaries of five instructions in the V Series instruction set that have the capacity to change the memory environment of a V Series system. Some of the material in this section may be affected by changes from one V Series system to another.

SECTION 5: I/O SUMMARY

This section describes I/O operations of the operating system, emphasizing logical I/O. The section presents an example, showing the sequence of events within the operating system as a user program opens and reads information from a disk file.

SECTION 6: MAINTENANCE PROCESSOR

This section briefly describes the Maintenance Processor and shows examples of Maintenance Processor functions that can be used during operating system support. This section contains hardware-dependent information that is related to a V 300 system. This information does not apply to other V Series systems.

SECTION 7: TRACE

This section describes a low-level trace facility included in the fault handler module of the MCP/VS operating system. Procedures for using this facility are briefly outlined. Some of the material in this section may be affected by changes from one V Series system to another.

SECTION 8: TRAK

This section briefly describes the TRAK feature of the operating system and includes a partial list of TRAK codes already embedded in the operating system.

SECTION 9: REMOTE SUPPORT

This section summarizes some of the remote support capabilities available on V Series systems. Some of the material in this section may be affected by changes from one V Series system to another.

APPENDIX A: OTHER SYSTEM INFORMATION

This appendix contains hardware-related system information (instruction set table, sub-MCP base memory assignments and so forth), software-related information (BCT summary, pre-term codes and so forth) and information about the LSTMOD utility. This section contains hardware-dependent information that is related to a V 300 system. This information does not apply to other V Series systems.

RESULTS

This document should provide you with quick, easily-referenced access to important technical information about the operating system. After using this document, you should have the information you need or know where to go to find it.

RELATED PRODUCT INFORMATION

The following documents are referenced in different portions of this document. They contain information that is needed to support the MCP/VS operating system.

V Series MCP/VS Architecture Manual

V Series MCP/VS Programming Reference Manual

V Series MCP/VS System Software Operations Guide, Volume 1: Installation

V Series MCP/VS System Software Operations Guide, Volume 2: System Commands

V Series MCP/VS System Software Operations Guide, Volume 3: System Utilities

V Series MCP/VS System Software Operations Guide, Volume 4: Output Messages

V Series Program Interfaces Programming Reference Manual

V 300 System Reference Manual

V 300 Maintenance Users Guide (System Design Specification 1997 5432)

V Series Instruction Set (System Design Specification 1997 5390)

SECTION 1
OPERATING SYSTEM MODULES

MCP/VS is an operating system developed for Unisys V Series systems. It is an advanced operating system that incorporates many concepts and features that are new to Unisys products. At the same time, it provides object code compatibility with previous Unisys B 2000/B 3000/B 4000 Series systems.

The 2.0 release of MCP/VS is composed of many separate modules that are linked together to form the operating system. A detailed description of the V Series architecture as implemented in MCP/VS 2.0 is provided in the *V Series MCP/VS Architecture Manual*. Table 1-1 lists each module of MCP/VS, summarizes the module's functions and lists the major data structures that the module operates on.

Modules and Overlays

Throughout this document, the term module refers to a part or component of the operating system. It includes both modules and overlays. There are differences between modules and overlays (relating to how long they remain in memory after they are loaded), but we use the single term module to emphasize the logical similarity between the two, rather than the operational differences.

TABLE OF OPERATING SYSTEM MODULES

Table 1-1. Operating System Modules—MCP/VS

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
ALC	Allocate Record Processing	Processes Allocate record. Called from control instruction processing module (CCD).	-EU Table
BIO	BNA Logical I/O Initiator	Handles logical I/O functions for BNA HOST SERVICES. Communicates with BNA Cooperator on remote system. Opens and closes port for information transfer. Processes opens, reads, writes and closes for remote files.	-File Information Block (FIB) -File Information Area (FIA) including external FIB -IOAT Tables (soft IOAT)
BOJ	Beginning of Job	Completes beginning of job processing for all tasks (except IRs). Called only from the BOTSK, INTBOT or INTBTI routines in the GLB module. Also initiates beginning of job processing for a DMSII (DBP) task.	-Mix Table -Program Name Table
BOO	Bootstrap	Loads the modules needed by system initialization module into memory. Invoked by the system loader (via the main entry point in the MCP code file) during coldstart or halt/load.	-None
C2C	Core to Core	Handles all user Core to Core BCTs (BCT 414). Performs synchronous data transfer between user programs. Notifies other modules when CRCR transfers are part of Complex Wait, Doze, Datacomm Wait, and Handler Acquire Next BCTs.	-Mix Table -User Code Memory Area (for both sending and receiving programs)
CAL	Program Call	Handles all processing resulting from the program call BCT (BCT 794) for both batch and timesharing jobs. Frequently used to call bound intrinsic program SORT:	-MCP Data Page of the task issuing program call BCT (reads Page Information) -Mix Table
CCD	Control Command Input Processing	Initiates and completes processing of MCP control instructions. Called by COMNIR independent runner or from the BOTSK / INTBOT / INTBTI routines in the GLB module.	-Mix Table

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
CC1	Control Command Handler Number 1	Called only by CCD module when CCD is running for the COMNIR independent runner or for a user task. Processes the following MCP control instructions: BCL BINARY BUFFER CHARGE DATA END ENDCTL ENDAT LOCK MEMDUMP MEMORY PRIORITY QWKMEM QWKPOOL RERUN TEST	-Mix Table
CC2	Control Command Handler Number 2	Called only by CCD module when CCD is running for the COMNIR independent runner or for a user task. Processes the following MCP control instructions: AFTER COMPILE DEBUG EXECUTE INSERT VALUE	-Mix Table
CFM3	Configuration File Maintenance	Accesses and maintains system configuration file used by running system. File stored on disk. Allows open, close, read and write of configuration file.	-System Configuration File (running system version on disk. CFM does *not* read configuration file on *flexible* disk. Flexible disk version only accessed by CONFIG utility.)

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
CLS	Close File	Closes an open file. Handles close retain and close with release (which returns any closed devices to system use).	-File Information Block (internal FIB) -File Information Area (including external FIB) -IOAT Tables (hard & soft IOATs) -Disk File Headers (stored in memory -Mix Table -EU Table
CND	Command Processing	Contains code executed as COMNIR independent runner. COMNIR is a driver routine. It is called by other parts of the operating system and initiates tasks for them.	
CPG	Compile & Go Processing	Initiates beginning of the job processing for the Go phase of a Compile-and-Go. Executes program just compiled. Only called from the BOTSK routine in GLB module.	-Mix Table -Reinstate List Entry (read)
CPS	Pass File Generator	Generates pass file used by an intrinsic program (SYSTEM/COPY, DMPALL and so forth). Performs enough syntax checking to construct the pass file. Primary syntax checking resides in intrinsic. Only called by BOTSK routine in GLB module. Calls ITN to schedule intrinsic.	-Mix Table
CSL	Installation Label Record Processing	Processes the Installation Label (LABEL1) record. Tape only.	-Installation Label Table -Status Template Blocks
CWT	Complex Wait	Handles all user Complex Wait BCTs (BCT 994). User tasks are suspended until one of the events specified in the BCT occurs.	-Mix Table -Complex Wait Table -User Code Memory Area (of waiting programs)
DBG	Debug	Handles processing for Interactive DEBUG. Invoked by keyboard commands Provides: -Single instruction processing -Stopping on breakpoints -Traces of user programs and the operating system	-Reinstate List Entries (read) -MCP Data Page of task being debugged (reads BCT Table) -Mix Table -IOAT Tables (hard & soft IOATs)

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
DCM	Batch/Datacomm Timesharing	Handles DataComm Read and Write BCTs (BCT 354). Transfers data between user tasks that read and write to datacomm (example EDITOR programs) and handler programs that pass information to the MCS (example TSHFEP).	-Mix Table -IOAT Tables (hard IOATs)
DCP	DCP I/O Driver	Communicates with MCS and Unisys V Series DCPs, (B874, B974, ORS DLP, and Telcom DLP).	-IOAT Tables (hard IOATs) -DCP Tables (MCS Table, DCP Table, Station Table) -MCS Buffer (contains soft IOAT for DCP file opened by MCS.
DCU	DCP Initialization	Builds DCP tables and loads DCP module when DCP option is set. Processes MCPNpF file (built by CPNDL1, NDL974 or NIFMRG). Calls intrinsic (B974LD or LDHOST) to load firmware to DCP. Unloads module when option is reset.	-IOAT Tables (hard IOATs) -DCP Tables (MCS Table, DCP Table, Station Table)
DFS	Disk File Security	Assigns security to and retrieves security from all disk and diskpack files. Checks open and close of secured files and overwrites all sensitive data files at remove time.	-Mix Table
DIO	Direct I/O	Handles Direct I/O user interface processing. Provides open, close, read and write services for direct I/O files.	-Direct I/O FIB -I/O Queue Elements -Mix Table -EU Table -IOAT Tables (hard IOATs)
DKM	Disk Available Table Maintenance	Allocates and deallocates space on all disk (100-byte) media. Manages all maintenance of disk available table (except for SQU, DSK and SIN modules).	-EU Table -Disk Available Table -Disk Master Available Table
DLP	DLP Record Processing	Processes DLP record. Builds channel table entry. Links channel R/D area into R/D chain. Builds hard IOAT for the SSP.	-Channel R/D Area -Channel Tables -IOAT Tables (hard IOAT for SSP)

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
DMS	DMS II	Handles all DMS II and multi-user ISAM BCT's (BCT 894 and so forth).	-Database User Program Control Table -Database Program (DBP) Table -Mix Table
DPM	Diskpack Maintenance	Handles allocation and deallocation of diskpack space. Manages file headers for all files on single or multiple diskpacks.	-Diskpack Directories -Diskpack Available Tables -Diskpack Master Available Tables -Diskpack Labels -Diskpack File Headers -Mix Table -EU Table
DRM	Disk Directory Maintenance	Provides access to and maintains disk directory for all disk (100-byte) media. Maintains file names, file in use flags. Adds and removes files from directory. Called by OPN, CLS, BOJ, EOJ and others.	-EU Table -Disk Directory -Disk File Headers (stored in memory and on disk)
DRV	OCS Driver	Contains code used by OCSDRV independent runner. Drives OCSs. Calculates wake up time for each OCS. Wakes up each OCS when automatic display is scheduled.	-AD Rule Table -Mix Table -IOAT Tables (hard IOAT) -EU Table -SPO Tank (stored in memory)
DSK	Disk Record Processing	Parses DISK, PACK and MOREDISK records in system configuration file. Builds corresponding EU table entries. Builds disk or diskpack label. Builds disk or diskpack available table entry and master available table entry. Processes DISK or PACK commands (peripherals added while system is running). Called by status routines the 1st time a disk becomes ready.	-EU Table -Disk Subsystem Table

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
FAU	Fault Handler	Called by Hardware Call procedure. Handles all tasks that receive fault indicators (processor faults, soft faults, and so forth). Determines severity of fault and responds appropriately. Displays Fault Screen if necessary	-Fault Indicators -Stack (Hardware Call stack frame) -Mix Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read)
FHS	Fault Support	Gives Fault Handler addressability to needed memory areas.	Not Applicable
FUN	Function BCT Processing	-Handles all user Function BCTs (BCT 534), including: ZIPSPO, CHGSEC, GETMCP, EQUATE, JOBINF and STATUS. -Handles user ZIP BCTs (BCT 274) and SPO-MESSAGE BCTs (BCT 474). -Called from SRT module to provide label equate function for Sort BCTs (BCT 254/255).	-Mix Table
GDR	Get Disk / Diskpack Directory BCT Processing (BCT 214)	Reads disk or diskpack directory and creates work file containing file IDs. Invoked by BCT 0214 (variant B or C). Used primarily by SYSTEM/COPY.	-Disk Directory -Diskpack Directory
GLB	Global	Serves as a central location for all short routines that must reside in memory. Loads and unloads segments in the overlay area. Loads and unloads modules. Creates and terminates independent runners. Contains entry points for most BCTs. Contains beginning of task routines for user tasks and intrinsic programs.	-Mix Table -EU Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read) -Program Name Table

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
IOM	Physical I/O	Handles low level Input/Output services, including: -Translation of virtual I/O descriptors to physical I/O descriptors -Translation of physical result descriptors to virtual result descriptors -I/O queueing, dequeueing and firing -I/O complete handling -I/O error recovery.	-I/O Queue Elements (system queue elements and user queue elements -Channel R/D Area -Exchange Queues -Mix Table -EU Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read) -Block Lockout Table
ISC	Inter System Control	Provides file-level interface to Inter System Control (ISC) DLP. Receives data from and returns data to LIO module. Handles most of open and close for ISC files.	-ISC External Buffer -Mix Table -IOAT Tables (hard IOATs) -I/O Queue Elements (user queue elements)
ITN	Intrinsic Scheduling	Schedules intrinsic programs. Handles most of BOJ for intrinsic tasks. Called when intrinsic task has been created, but needs schedule initiation.	-Intrinsic Tables -Mix Table
KBD	Keyboard Input Processing	Handles all commands entered through the keyboard. Passes MCP control instructions to Control Command (CCD) module. Processes keyboard input messages and calls appropriate module.	-Mix Table -Reinstate List Entry (read) -Block Lockout Table
KBF	Keyboard Processing (BF, RF Commands)	Processes following keyboard commands: -BF (Display backup files) -RF (Remove backup files) backup file type can be PRN, PCH, or DMP.	-IOAT Tables (hard & soft IOATs)
KBM	Keyboard Output Processing	Displays all output messages on ODT. Writes output messages in ODT log if SLOG option is set.	-Mix Table -EU Table
KBN	BNA Support (Keyboard Processing)	Processes BNA keyboard commands AT, HN, NET, and NW. Handles BNA phase change BCT (BCT 774) and MCPMSG receive BCT (BCT 754). Handles AT and ENDAT control instructions.	-Port BNA Global Storage Area -Mix Table

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
KCD	Keyboard Processing (CD, RD Commands)	Processes following keyboard commands: -CD (Display inactive pseudo card decks on disk) -RD (Remove inactive pseudo card decks on disk).	
KDB	Keyboard Processing (DB Command)	Processes following keyboard commands: -DBMM (Set/reset maintenance mode) -DBCA (Clear SSP address) -DBUA (Unlock address) -DBUL (Clear locked files on shared systems) -DBIC (Initialize controller parameters) Calls other modules to process following commands: -DBDR (Produce SSP content report) Calls KSM. -DBSR (Produce block lockout table report) Calls KSM. -DBCL (Cancel DLP) Calls KDS.	-EU Table -IOAT Tables (hard & soft IOATs)
KDC	Keyboard Processing (DC Command and Others)	Processes following keyboard commands: -CK (Test peripheral device) -DC (Display compilation data) -FN (List file names)	-IOAT Tables (hard & soft IOATs)
KDL	Keyboard Processing (DL Command and Others)	Processes following keyboard commands: -DL (Delete peripheral devices) -RW (Rewind and unload tape) -SV (Save a peripheral unit) -UR (Inhibit or uninhibit unit)-XC (Inhibit, uninhibit or display status of a DLP)	-EU Table -IOAT Tables (hard & soft IOATs)
KDQ	Keyboard Processing (DQ Command)	Passes all or part of the ODT Log to an OCS. Information passed either in reply to DQ command, or on a cyclic basis (for automatic screen display).	

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
KDS	Keyboard Processing (DM Command and Others)	Processes following keyboard commands: -DM (Dump and continue) -DP (Dump and discontinue) -DS (Discontinue program) -FP (Flush program) -GO (Resume stopped program) -LP (Lock program) -NL (Resume as new location) -ST (Suspend program processing) -UP (Unlock program)	-Mix Table -EU Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read) -Program Name Table -Block Lockout Table
KER	Kernel	Dispatches all other tasks. Invoked after any interrupt occurs on the system (example: I/O complete, Lock/Event, and so forth). Issues BRV instruction to pass control to task being reinstated.	-Reinstate List Entries -Hardware parameters provided with interrupt -Interrupt Mask
KIL	Keyboard Processing (IL Command and Others)	Processes following keyboard commands: -CN (Display tape number) -FI (List files in use by a program) -FR (Designate final reel) -IL (Assign labeled file) -PR (Change priority) -UL (Assign unlabeled file)	-Mix Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read)
KIN	Keyboard Processing (IN Command and Others)	Processes following keyboard commands: -AX (Respond to an ACCEPT) -IN (Insert data into program) -OT (Display program data) -SK (Skip backup records) -SP (Display/set/change database parameters) -SW (Set program switches)	-Mix Table
KKS	Keyboard Processing (KS Command and Others)	Processes following keyboard commands: -KA (Analyze disk directory) -KP (Print disk segments) -KS (Analyze disk space) -SQ (Squash disk) -SQP (Squash diskpack)	-Mix Table
KKX	Keyboard Processing (KX Command and AUTO KX)	Processes KX keyboard command. Handles auto-KX request from DKM module. Looks for files of the correct size and allows operator to remove them. Usually executed to permit the system to continue running.	-EU Table -IOAT Tables (hard & soft IOATs)

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
KLH	Keyboard Processing (LH Command and Others)	Processes following keyboard commands: -LH (Load firmware into a channel controller or DCP) -RY (Ready peripheral device) Provides following functions: -Train printer load (called by status routine) -Train printer buffer load -Automatic load host-B974 Load Host (EX B974LD)	-EU Table -IOAT Tables (hard IOATs)
KLN	Keyboard Processing (LN Command and Others)	Processes following keyboard commands: -LC (Insert log comment) -LN (Transfer and print log) -TL (Transfer log)	
KOK	Keyboard Processing (OK Command)	Processes following keyboard commands: -OF (Indicate optional file) -OK (Continue processing suspended program) -PO (Power off diskpack)	-Mix Table -IOAT Tables (hard & soft IOATs)
KOL	Keyboard Processing (OL Command)	Processes following keyboard command: -OL (Display peripheral status)	-EU Table -IOAT Tables (hard & soft IOATs)
KPB	Keyboard Processing (PB Command and Others)	Processes following keyboard commands: -CV (Convert punch backup file to pseudo card deck) -IR (Initiate DMSII recovery) -LD (Create pseudo card deck) -PB (Print backup file - using PBDOUT) -PC (Create punch backup file) -PM (Print memory dump)	-Mix Table
KPD	Keyboard Processing (PD Command)	Processes following keyboard command: -PD (Print Directory)	
KPG	Keyboard Processing (PG Command and Others)	Processes following keyboard commands: -AC (Place tape number in label) -PG (Purge magnetic tape) -RP (Ready and purge magnetic tape) -SN (Place tape number in label) -TM (Write tape mark) Provides purge tape function routine, called by the status routines.	-Mix Table -IOAT Tables (hard & soft IOATs)

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
KPS	Keyboard Processing (PS Command and Others)	Processes following keyboard commands: -PA (Analyze diskpack directory) -PP (Print diskpack sectors) -PS (Analyze diskpack space)	-Mix Table
KQT	Keyboard Processing (QT Command)	Processes following keyboard command: -QT (Quit program operation)	-Mix Table
KRM	Keyboard Processing (RM and CH Commands)	Processes following commands: -CHANGE / CH (Change file identifier) -REMOVE / RM (Remove file)	
KRN	Keyboard Processing (RN Command and Others)	Processes following keyboard commands: -DA (Deactivate pseudo reader) -RN (Activate pseudo reader) -SD (Set deck limit)	
KRS	Keyboard Processing (RS Command and Others)	Processes following keyboard commands: -MR (Remove duplicate file -new) -RA (Remove after linkage) -RM (Remove duplicate file - old) -RS (Remove job from schedule) -SB (Interrogate database activity) -WS (Display jobs in schedule)	-Mix Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read) -Program Name Table
KSM	System Maintenance Commands	Processes following keyboard commands: -ALTER (Change system parameters) -DD (Delete DLP) -DX (Display DLPs attached to an exchange) -SHOW (Display system parameters) -XA (Add/remove a channel from exchange) -HL (Request halt/load) called from KBD. Auto H/L called from FAU module.	-Mix Table -EU Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read) -Block Lockout Table -Channel Table
KSS	Keyboard Processing (MCS Support Commands)	Processes following keyboard commands: -BO (Blackout access code) -SI (Set remote device identification) -SM (Send system messages) -SS (Send message) -TI (Type processing time)	-Mix Table -Reinstate List Entry (read)

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
KTO	Keyboard Processing (TO Command and Others)	Processes following keyboard commands: -FM (Direct file to device - special forms response) -OU (Direct file to device) -RO (Reset system option) -SO (Set system option) -TO (Display system option status)	-Mix Table -Reinstate List Entry (read)
KWT	Keyboard Processing (WT Command and Others)	Processes following keyboard commands: -DR (Change date) -RQ (Remove STOQ entry) -TR (Change time) -WD (Display date) -WM (Display MCP version) -WO (Display active QWIK operation count) -WQ (Display STOQ count) -WT (Display time)	
KWX	Keyboard Processing (WX Command and Others)	Processes following keyboard commands: -RX (Return deleted space) -WC (Display available memory) -WX (Display removed space) -XD (Remove disk segments) -XM (Remove memory from system)	-Mix Table
KWY	Keyboard Processing (WY Command and Others)	Processes following keyboard commands: -AJ (Display active jobs) -MX (Display jobs in mix) -WJ (Display waiting jobs) -WY (Display job status)	-Mix Table -EU Table -IOAT Tables (hard & soft IOATs)
KXP	Keyboard Processing (XP Command and Others)	Processes following keyboard commands: -RXP (Return Deleted Diskpack Space) -WXP (Display Deleted Diskpack Space) -XP (Remove Diskpack Sectors)	-Diskpack Master Available Table -Diskpack Working Available Table
LEQ	User File / Label Equate Processing	Handles: -File/Label equates executed by user -Security clause in an MCP control instruction -Start and Stop MCP control instructions	-Program Parameter Block -Mix Table

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
LIO	Logical I/O	Handles I/O service requests between user tasks (or IRs) and IOM module. Provides tailored I/O paths for different media. Dispatches physical I/Os for efficient handling of I/O buffers. Processes BCTs or HCLs for Read, Write, Seek, Position, and User Overlay.	-I/O Queue Elements (user queue elements and system queue elements) -File Information Block (FIB) -File Information Area (FIA) including external FIB -IOAT Tables (hard & soft IOATs) -Mix Table -Block Lockout Table
MCR	Magnetic Ink Character Recognition (MICR)	Handles all real time I/O for reader/sorter devices (BCT 374 processing).	-Mix Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read) -Program Name Table -MICR Memory Map
MES	Message Module	Handles inter-task communication for the operating system. Coordinates message handling for all tasks. Consistently handles tasks waiting on single or multiple events.	-Reinstate List Entry (read) -Internal (Task Wait Table, Task Waiting Lists)
MLG	Maintenance Log	Creates maintenance log files and generates all MLOG records. Always loaded by system initialization. Invoked by system initialization, I/O error, I/O complete, various control instructions and keyboard commands. Receives information for MLOG in parameters from callers.	-IOAT Tables (hard & soft IOATs) -EU Table -File Information Block (FIB) -Mix Table
MMG	Memory Manager	-Allocates and deallocates memory to the operating system and all user jobs running in the mix. -Dynamically changes placement of tasks in memory to optimize memory usage. -Manages roll out of information from memory to disk and roll in from disk to memory.	-Stack -Reinstate List -Environment Tables -Memory Area Status Tables -Memory Area Table -Mix Table -Block Lockout Table -Shared Area Table

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
OCS	OCS Processing	Places response to OCS keyboard input in the screen buffer. Formats automatic screen update and places in screen buffer. Routes keyboard input command to KBD module. Processes AD keyboard command and AD text on an OCS/ODT unit record.	-AD Rule Table -Mix Table -IOAT Tables (hard IOAT) -EU Table -SPO Tank (stored in memory)
OIO	OCS I/O	Contains code executed by OCSDRV independent runner. Handles I/Os from screen buffer to OCS.	-IOAT Tables (hard IOATs)
OPN	Open File	Handles normal OPEN BCT (BCT 134) and OPEN AVAILABLE BCT (BCT 674). Assigns requested file to requesting task. On file opened input, checks for existing file. If found on normal open or open available, appropriate system tables are updated and task is reinstated. If not found on normal open, message displayed, operator intervention may be required. (A task is never suspended on an OPEN AVAILABLE BCT.) On file opened output, searches for available resources.	-Mix Table -File Information Area (FIA) including external FIB and disk file headers stored in memory. -IOAT Tables (hard & soft IOATs) -EU Table -File Information Block (FIB)
PAD	PATCH and DISPLAY Processing	Processes MCP control instructions PATCH, DISPLAY and WHATS. Displays or modifies data in files.	-EU Table
PCR	Disk Pseudo Card Reader Processing	Maintains Disk (100-byte) PCR directory. Executes PCR decks (RN =,RN #). Called by: -change routines when a file is changed to a pseudo reader (#00000). -OCSIO to process RN and SD keyboard commands. -User Task when PCR is closed by an invalid card, but processing must continue for the remainder of the PCR.	

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
PRP	Diskpack Pseudo Card Reader Processing	Manages diskpack PCR directory. Executes PCR decks (RN =,RN #). Called by: -change routines when a file is changed to a pseudo reader (#00000). -OCSIO to process RN and SD keyboard commands. -User Task when PCR is closed by an invalid card, but processing must continue for the remainder of the PCR.	
PRT	Port File Functions	Processes following BCTs: OPEN(BCT 614) CLOSE(BCT 614 & 694) READ(BCT 614) WRITE(BCT 614) SET ATTRIBUTE(BCT 634) GET ATTRIBUTE(BCT 654) PLM SUPPORT(BCT 714) Only valid for BNA	-Mix Table -File Information Area (FIA) including: FIZ, external FIB, hardware type and Port Attribute Structure -Port Global Data Memory Area -Shared Subport Memory Area
PTM	Pre-Terminate	Called by other operating system modules when irrecoverable error is encountered. Pre-term code provided in call. For ARMed tasks, reinstates task at ARM address. For non-ARMed tasks, sets up error message based on code. Displays error message, DS or DP message, and suspends task.	-Stack (on MCP Data Page of task) -Mix Table -EU Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read)
PUP	PACKUP Record Processing	Parses LOAD, DUMP, ADD, UNLOAD, and CHECK syntax that specify diskpack. Creates a pass file. Executes a user program called PACKUP. (PACKUP bound intrinsic no longer supported).	-Mix Table
RES	Memory Resource Manager	Runs as independent runner. Handles tasks that are waiting to expand their memory or waiting to roll in information from disk to memory. Maintains tables of waiting tasks and each task's memory priority. Receives notification of waiting tasks through message module. Calls memory manager attempting to gain memory for tasks based on priority.	-Internal Tables -Reinstate List Entry (read)

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
RJE	Remote Job Entry	Supports Remote Job Entry (RJE) and other handler programs (like WFL). Loaded at system initialization and never unloaded. Sets and resets RJE option. Creates RJE handler. Processes EOJ for all handlers (WFL, BNA and so forth). Called by DCP module, woken by IOM, CLS, TRM and others.	-Mix Table -IOAT Tables (hard IOATs) -MCS Table
RLG	Run Log	Creates all run log files and generates all run log records. Loaded at coldstart, if USE RLOG record present. Invoked by beginning of job, terminate, open, close, system initialization (during halt/load) and various keyboard commands.	-Mix Table -IOAT Table (hard & soft IOATs) -File Information Block (FIB)
RUN	RUN Command Processing	Processes the RUN and SHARE MCP control instructions. Initiates scheduling of a timesharing task. Initializes Shared Area Table (SAT) and the Processor Information Block (PIB).	-Mix Table -Shared Area Table -Processor Information Block
SCA	Subport Candidate	Manages Subport Candidate List for PRT module. Adds candidates to list, searches list for matched candidates, reports matches to PRT and removes candidates from list.	-File Information Area (FIA) including Port Attribute Structure -Candidate List Memory Area -Port Global Data Memory Area
SEC	System Access Security	Processes LI, LO and related system commands. Processes all Security BCTs (BCT 534). Controls access to all system resources except files. (File access controlled by DFS module.)	-Mix Table -USERFL File -USRCOM File -USRTBL Table -MCP Data Page
SIN	System Initialization	Performs all coldstart and halt/load initialization. Builds most system tables, using information from system configuration file. Creates independent Runners (IRs). Builds disk directory and other required disk files at coldstart. Calls DLP, UNT, DSK and CSL modules.	-Mix Table -EU Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read) -Program Name Table -Block Lockout Table
SQU	Disk Squash	Repositions files on a disk ID to create maximum available contiguous space on that ID.	-EU Table

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
SRT	Sort	Processes old Sort BCT (BCT 254) Provides convenient method of sorting records in a file based on any combination of up to 50 different non-overlapping keys.	-Mix Table
STQ	STOQUE Processing	Handles STOQ (Storage Queue) BCTs (BCT 494). Transfers data between programs asynchronously. Notifies CWT module when STOQ transfers are part of Complex Wait BCTs.	-Mix Table -Complex Wait Table -User Code Memory Area (for both sending and receiving programs)
STS	System Status	Contains most of the routines that periodically maintain status of tasks and peripherals. Calls periodic log functions. Monitors MCP QWIK and calls User QWIK status routines. Includes unit status, disk status, and overtime task status routines. Contains code executed as Overtime I/O independent runner and midnight status independent runner.	-IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read) -Mix Table -EU Table
SYS	Syst (FLAME Support)	Performs BCT 554 connect. Causes MODLRD to load SYST module and execute IGNITE, the initialization routine of the FLAME package.	-Mix Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read) -Block Lockout Table
TCL	Terminate File Close	Closes all files when a task is terminated for normal or abnormal reasons. Forces a close with release (returns system resources for other use).	-File Information Block (internal FIB) -File Information Area (including external FIB) -IOAT Tables (hard & soft IOATs) -Disk File Headers (stored in memory -Mix Table -EU Table
TMD	Time and Date	Handles most variants of the Time and Date BCT (BCT 214).	-Mix Table

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
TMR	Time Event Handling	Receives timed wakeup conditions from WTG module. Notifies MSG module when timed wakeup condition is satisfied. Deletes timed wakeup conditions when requested by MSG module. Contains code executed as TIMRIR independent runner.	-Timer Table
TRC	User Program Trace	Handles tracing of user programs (not including the operating system when it is running for the program). Uses same method as debug module to format and output trace information for printer or printer backup. Runs entirely for task being traced. GT and NT are handled as asynchronous commands.	-Reinstate List Entries (read) -MCP Data Page of task being traced (Trace Data Storage and BCT Table) -Mix Table -IOAT Tables (hard & soft IOATs)
TRK	TRAK Support	Handles placement of TRAK information in re-circulating TRAK buffer. Called by other parts of operating system to provide debugging information. Functions only when TRAK option is set, otherwise calls to TRAK result in immediate return. Allows TRAK buffer to be processed into a file.	-TRAK Buffer

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
TRM	Terminate Task	Performs all processing necessary to terminate a task, including: -terminates secondary tasks linked to the terminated task (example: sorts, tasks initiated through timesharing and so forth) -processes exception conditions encountered during termination (such as open files, WFL, BNA, compilers, TSHMCS, program call, waiting STOQ) -cleans QWIK buffer, if necessary -receives all messages waiting for the terminated task -closes program code file -handles AFTER linkages depending on AFTR option -terminates programs ZIPed if task is a WFL-type task -calls CPG module for compile-and-go tasks -Logs task EOJ -delinks/clears mix entry and program name table -interrupts to kernel	-Mix Table -Active Job Table -IOAT Tables (hard & soft IOATs) -Reinstate List Entry (read) -Program Name Table -Shared Area Table
UNT	UNIT Record Processing	-Parses each UNIT record in system configuration file and builds corresponding IOATs (Procedure name = UNTCRD) -Parses CC/U, family name or ID nnn syntax. Returns pointers and other information about the requested device to caller. (Procedure name = GETCHU)	-IOAT Tables (hard IOATs)
UQK	User QWKMEM Processing	Reads user program overlay into program overlay area and moves overlay into overlay pool memory. Subsequent calls for that overlay become memory to memory moves, instead of physical I/Os. Reduces I/Os and overhead for programs that frequently use Overlay BCT (BCT 174).	-Mix Table -IOAT Tables (hard & soft IOATs)

Table 1-1. Operating System Modules—MCP/VS (Continued)

Module Mnemonic	Module Name	Module Function	Major Data Structures Operated On By Module
WTG	Waiting Conditions	Causes a task to wait for satisfaction of specific conditions (for example, Waiting No File, Waiting Memory). Called from various modules including OPN, CLS, DCM and others. Calls Time Event Handling module (TMR) to set up timed wait conditions. Calls Message module (MES) to suspend task. MES wakes up task if conditions are satisfied. Called from TMR and other modules to retry a function.	-Mix Table -EU Table -IOAT Tables (hard & soft IOATs)

TABLE OF INDEPENDENT RUNNERS

The MCP/VS operating system is designed to run for user tasks, performing different services as requested. In some situations, the operating system must perform services for itself, rather than any specific user task. When this happens, a portion of the operating system's code is executed as a self-contained unit, similar to a user task. The self-contained unit then provides or requests other portions of the operating system to provide the necessary services.

This kind of self-contained unit is called an Independent Runner (IR). IRs are very similar to user tasks, except that they are called into existence by the operating system itself, they have a higher priority than user tasks, and they can communicate with the rest of the operating system in ways that user tasks cannot.

IRs handle reporting, housekeeping and system status functions for the operating system. Some of the IRs execute periodically to help the system run more efficiently. Table 1-2 lists the most common independent runners, summarizes their functions and lists the major data structures they operate on.

Table 1-2. Independent Runners - MCP/VS

IR Mnemonic	IR Name	IR Function	Major Data Structures Operated On By IR
B874IR	B874 Read	Performs default read of B874 to transfer data to operating system. One of these exists for each B874 DCP on the system. Handles protocol between host system and the DCP.	-IOAT Tables (hard IOATs) -DCP Tables (MCS Table, DCP Table, Station Table) -Channel R/D Area
B974IR	B974 Read	Performs default read of B974 to transfer data to operating system. One of these exists for each B974, Telcom DLP, or ORS DLP on the system. Handles protocol between host system and the DCP.	-IOAT Tables (hard IOATs) -DCP Tables (MCS Table, DCP Table, Station Table) -Channel R/D Area
COMNIR	Command Processing IR	Handles commands entered through OCS or ODT. Passes command to KBD, passes response back to OCS or ODT. Executes code contained in CND module.	-IOAT Tables (hard IOATs) -AD Rule Table -MCP Data Page
DBG-IR	Debug OCS Driver	Handles communication with user at OCS or ODT. Formats debug screens and passes debug commands to debug module. Write trace information to printer or printer backup.	-Internal

Table 1-2. Independent Runners - MCP/VS (Continued)

IR Mnemonic	IR Name	IR Function	Major Data Structures Operated On By IR
IDLE	Idle CPU Absorption	Dispatched by the kernel when no other tasks are ready to run. Absorbs all idle CPU time. Branches to itself in endless loop.	
I/OERR	I/O Error Processing	Handles processing of all I/O completes that have exception conditions. Handles all I/O retries that do not require waiting.	-I/O Queue Elements (system queue elements and user queue elements) -Channel R/D Area -Exchange Queues
IOT	Normal I/O Complete	Handles all normal I/O completes (those with no exception conditions).	-I/O Queue Elements (system queue elements and user queue elements) -Channel R/D Area -Exchange Queues
JOExxx	Supplemental I/O Error Processing	Identical IRs that handle I/O completes with exception conditions passed off from I/OERR. Usually these are conditions that require waiting or have possibilities for I/O failure within themselves. There are always at least two IRs. More than two IRs may exist if the need arises. These IRs work from a single exchange queue (called the Joe queue). One IR may handle error conditions encountered by another.	-I/O Queue Elements (system queue elements and user queue elements) -Channel R/D Area -Exchange Queues
KILLME	Terminate Task Processing	Called when task has gone to EOJ. Calls memory manager to return task's memory for other use.	-None
OCSDRV	OCS Driver	Awakens periodically (based on AD update interval). Cancels default reads of ODTOCS or UNIOCS independent runners. IRs then update their screens and then return to default reads.	-IOAT Tables (hard IOATs) -OCS Table
ODTOCS	ODT Reader	Performs default reads of ODT to transfer data to the operating system. Updates ODT screen when default read is canceled by OCSDRV independent runner.	

Table 1-2. Independent Runners - MCP/VS (Continued)

IR Mnemonic	IR Name	IR Function	Major Data Structures Operated On By IR
RCSRCE	Memory Resource Manager	Handles tasks that are waiting to expand their memory or waiting to roll in information from disk to memory. Maintains tables of waiting tasks and each task's memory priority. Receives notification of waiting tasks through message Module. Calls memory manager attempting to gain memory for tasks based on priority.	-Internal
RIDRVR	Roll In Driver	Handles transfer of information from disk into memory. Calls I/O module and disk maintenance.	-Internal
RLGTIR	Run Log Transfer	Closes existing run log file with pseudo-EOJ record. Opens new run log file with pseudo-BOJ record.	-Mix Table
RODRVR	Roll Out Driver	Handles transfer of information from memory out to disk. Calls I/O module, disk maintenance and disk directory maintenance.	-Internal
STATDM	DMSII Status	Handles processing related to the DMSII status period (set by LIMIT DMSTAT record).	-Database Control Table -DBP State Entry Table
STAT-1	Status Routines	Initiates periodic status functions for operating system. Handles processing for functions started because of changes discovered during periodic checks (diskpack recovery, tape label recognition, output device label creation and so forth).	
STKOIR	Stack Overflow	Expands stack for a task that has run out of stack. Calls MMG to expand stack.	
TIMRIR	Timer IR	Handles periodic waking of tasks that are waiting on some condition.	
TRC-IR	User Program Trace Write	Opens, writes and closes printer backup files containing user program traces.	
UNIOCS	Uniline DLP Driver	Performs constant default reads of Uniline DLP to transfer data to the operating system. Updates screen when default read is canceled by OCSDRV independent runner.	

Table 1-2. Independent Runners - MCP/VS (Continued)

IR Mnemonic	IR Name	IR Function	Major Data Structures Operated On By IR
Various Names	Device Status Functions	Status independent runner (STAT-1) creates separate independent runners to perform several device status functions. IRs created when device changes state from not ready to ready. IR name indicates device. Functions that require separate IRs include diskpack recovery, tape label recognition and other functions that are both complex and transient.	

COMMAND / MODULE CROSS REFERENCE TABLE

Table 1-3 lists the commands of the MCP/VS operating system and the operating system module that processes each command.

Table 1-3. Command/Module Cross Reference

Command	Module
AC	KPG
AD	OCS
ADD	CPS
AFTER	CC2
AJ	KWY
AL	CCD
ALLOCATE	ALC
ALTER	KSM
AT	KBN
AX	KIN
BCL/DATAB	CC1
BEGINUSER	SEC
BF	KBF
BINARY	CC1
BO	KSS
BT	TRK
BUFFER	CC1
CA	GLB
CD	KCD
CHANGE/CH	KRM
CHARGE/CG	CC1
CHECK	CPS
CK	KDC
CN	KIL
COMPARE	CPS
COMPILE/CMP	CC2
COPY	CPS
COPYADD	CPS
CV	KPB
DA	KRN
DATA/EBCDIC	CC1
DB	KDB
DC	KDC
DD	KSM
DEBUG	CC2

Table 1-3. Command/Module Cross Reference (Continued)

Command	Module
DIR	CPS
DISK	DSK
DISPLAY	PAD
DL	KDL
DLP	DLP
DM	KDS
DMCOPY	CPS
DP	KDS
DQ	KDQ
DR	KWT
DS	KDS
DUMP	CPS
DX	KSM
ED	DBG
END/ENDAT/ENDCTL	CC1
ENDUSER	SEC
ET	TRK
EXECUTE/EX	CC2
FI	KIL
FILE	LEQ
FM	KTO
FN	KDC
FP	KDS
FR	KIL
GENERATE/GEN	CPS
GO	KDS
GT	TRC
HL	GLB
HN	KBN
ID	DBG
IL	KIL
IN	KIN
INSERT/IN	CC2
IR	KPB
KA	KKS
KP	KKS
KS	KKS
KX	KKX
LABEL1	LBL

Table 1-3. Command/Module Cross Reference (Continued)

Command	Module
LC	KLN
LD	KPB
LH	KLH
LI	SEC
LN	KLN
LO	SEC
LOAD	CPS
LOCK	CC1
LP	KDS
MEM	CC1
MEMDUMP	CC1
MERGE	CPS
MOVE	CPS
MR	KRS
MX	KWY
NE	KBN
NL	KDS
NT	GLB
NW	KBN
OF	KOK
OK	KOK
OL	KOL
OT	KIN
OU	KTO
PA	KPS
PACK	DSK
PATCH	PAD
PB	KPB
PBDPRN	CPS
PC	KPB
PD	KPD
PERFORM/PFM	CPS
PG	KPG
PM	KPB
PO	KOK
PP	KPS
PR	KIL
PRIORITY/PR	CC1
PRP	CC1
PS	KPS

Table 1-3. Command/Module Cross Reference (Continued)

Command	Module
QD	DBG
QT	KQT
RA	KRS
RD	KCD
REMOVE/RM	KRM
RERUN	CC1
RF	KBF
RM	KRS
RN	KRN
RO	KTO
RP	KPG
RQ	KWT
RS	KRS
RUN	RUN
RW	KDL
RX	KWX
RY	KLH
SB	KRS
SD	KRN
SECURITY	LEQ
SH	CCD
SHOW	KSM
SI	KSS
SK	KIN
SM	KSS
SN	KPG
SO	KTO
SP	KIN
SPO	KBD
SQ	KKS
SS	KSS
ST	KDS
START	LEQ
STOP	LEQ
SV	KDL
SW	KIN
TEST	CC1
TI	KSS
TIME	CC1
TL	KLN

Table 1-3. Command/Module Cross Reference (Continued)

Command	Module
TM	KPG
TO	KTO
TR	KWT
TRACE	TRC
UL	KIL
UNIT	UNT
UNLOAD	CPS
UP	KDS
UR	KDL
USER	SEC
VALUE/VA	CC2
WB	ITN
WC	KWX
WD	KWT
WJ	KWY
WM	KWT
WO	KWT
WQ	KWT
WS	KRS
WT	KWT
WX	KWX
WY	KWY
XA	KSM
XC	KDL
XD	KWX
XM	KWX
XP	KXP

MODULE NAME TABLE

Table 1-4 lists the modules of the MCP/VS operating system and provides the following module names for each module:

- Module mnemonic. A three-character abbreviation for the full module name. This is the name used most frequently throughout this document.
- SPRITE name. This name is also known as the MID name because it is the name used in the Module Interface Description (MID) of the operating system.
- SPRASM name. Limited to six characters in length, this name is often used when entering module names for TRAK debugging (through the BT command).

Table 1-4. Module Names

Module Mnemonic	SPRITE Name (MID Name)	SPRASM Name (TRAK Name)
ALC	ccall	!CCALL
BIO	bnalio_ initiator	BNAMOD
BOJ	boj	BOJ
BOO	bootstrap	—
C2C	core_ to_ core_ mod	!CR2CR
CAL	procl	PROCL
CCD	ctlcd	!CTLCD
CC1	cc_ 01	!CC-01
CC2	cc_ 02	!CC-02
CFM	cf_ io	CF-IO
CLS	close_ mod	!CLOSE
CND	comnd	!COMND
CPG	cmpgo	!CMPGO
CPS	ccpas	!CCPA
CSL	cslbl	!CSLBL
CWT	complex_ wait	!CWT
DBG	debug_ module	DEBUG
DCM	dcom	!DCOM
DCP	dcp_ mod	DCP
DCU	dcpa	!DCPA
DFS	dfsec	!DFSEC
DIO	dirio	!DIRIO
DKM	disk_ maintenance	DKMMOD
DLP	cc_ dlp	!CCDLP
DMS	dms2	!DMS2 or DMS
DPM	pack_ maintenance	DPM
DRM	directory_ maintenance	DRMMOD
DRV	ocdrv	!OCDRV
DSK	ccdsk	!CCDSK

Table 1-4. Module Names (Continued)

Module Mnemonic	SPRITE Name (MID Name)	SPRASM Name (TRAK Name)
FAU	fault_ handler_ module	FAULT
FHS	faultsupport_ module	FLTSUP
FUN	funct	!FUNCT
GDR	gtdir	!GTDIR
GLB	global_ mod	GLOBAL
IOM	io_ mod	I/OMOD or IOMOD
ISC	isc_ mod	ISC
ITN	intrinsic_ segment	!INTRN
KBD	kbdin	!KBDIN
KBF	kb_ bf	!KB-BF
KBM	kbout_ mod	KBOMOD
KBN	bna_ support	!KB-BN
KCD	kb_ cd	!KB-CD
KDB	kb_ db	!KB-DB
KDC	kb_ dc	!KB-DC
KDL	kb_ dl	!KB-DL
KDQ	kb_ dq	!KB-DQ
KDS	kb_ ds	!KB-DS
KER	kernel_ mod	KERMOD
KIL	kb_ il	!KB-IL
KIN	kb_ in	!KB-IN
KKS	kb_ ks	!KB-KS
KKX	kb_ kx	!KB-KX
KLH	kb_ lh	!KB-LH
KLN	kb___ og	!KB-LN
KOK	kb_ ok	!KB-OK
KOL	kb_ ol	KB-OL
KPB	kb_ pb	!KB-PB
KPD	kb_ pd	!KB-PD
KPG	kb_ pg	!KB-PG
KPS	kb_ ps	!KB-PS
KQT	kb_ qt	!KB-QT
KRM	kb_ k2	!KB-K2
KRN	kb_ rn	!KB-RN
KRS	kb_ rs	!KB-RS
KSM	sysmaint_ commands	KBDSYS
KSS	kb_ ss	!KB-SS
KTO	kb_ to	!KB-TO
KWT	kb_ wt	!KB-WT
KWX	kb_ wx	!KB-WX
KWY	kb_ wy	!KB-WY
KXP	Kb_ xp	!KB-XP

Table 1-4. Module Names (Continued)

Module Mnemonic	SPRITE Name (MID Name)	SPRASM Name (TRAK Name)
LEQ	lbleq	!LBLEQ
LIO	liom	!LIOM
MCR	micr	!MICR
MES	message_ module	MESMOD
MLG	maint_ log	!MLOGT
MMGf	mem_ mgr	MMGR
OCS	ocsmod	OCSMOD
OIO	ocsio	!OCSIO
OPN	open_ mod	!OPEN
PAD	ccdsp	!CCDSP
PCR	pcr	!PCR
PRP	pcrpk	!PCRPK
PRT	port_ mod	—
PTM	prtrm	!PRTRM
PUP	cc_ pup	!CCPUP or PACKUP
RJE	rje_ mod	!RJE
RLG	run_ log	!RLOG
RSE	resmod	RESMOD
RUN	ccrun	!CCRUN
SCA	subport_ candidates	—
SEC	sys_ access_ security	!SYACS
SIN	system_ initialization	—
SQU	sqash	!SQASH
SRT	sort	!SORT
STQ	stoq_ mod	!STOQ
STS	status	STATUS;
SYS	syst	SYST;
TCL	trmcl	!TRMCL
TMD	timdt	!TIMDT
TMR	timmod	TIMMOD
TRC	trace_ module	TRCMOD
TRK	track_ mod	TRKMOD
TRM	term	!TERM
UNT	ccunt	!CCUNT
UQK	user_ quickmem_ mod	!UQWIK
WTG	wtgmod	WTGMOD

SECTION 2
THE LINKER LISTING

The modules of MCP/VS are linked together with a program named LINKER. As part of the linking process, LINKER produces a printer backup file containing information about the operating system. This listing is conceptually similar to the compile listing of a user program, but it contains more information than a compile listing and the information is more detailed and specific to the operating system.

The LINKER listing is available for Unisys support personnel along with source listings of the MCP/VS operating system. Used with the source listings, the LINKER listing can be a helpful tool.

This section describes:

- The contents of the LINKER listing. Much of the information in the listing is not commonly used during operating system support. This infrequently used information is described only briefly. The commonly-used support information is described in greater detail.
- How to find the available patch area in a module.
- How to translate environment numbers (displayed on the fault screen when the system fails) into the segment number and name of a module within the operating system.

CONTENTS OF THE LINKER LISTING

The LINKER produces a listing as it links a set of modules (ICMs and so forth) into an operating system code file. Some of the information in the listing comes from LINKER control language statements that control the LINKER, while other information is generated as the LINKER runs.

Beginning at the start of the listing, the contents of a LINKER listing are as follows:

COMMENTS

At the top of every page is a line showing the date and time that the operating system was linked together and the version of the LINKER used.

The LINKER control language is a set of instructions that tells the LINKER how to create a complete operating system. The date of the particular LINKER input file used is shown on comment lines on the first page of the listing.

This information can verify that you have the correct LINKER listing, but it is not commonly used during operating system support.

MODULE PRINT STATEMENTS

A set of module print statements begins on the first page, below the comments. These statements simply tell the LINKER to print out the source and object code of a module while it links the operating system together. The module print statements take up two to three pages because of the large number of modules that make up the operating system.

This information is not commonly used during operating system support.

MEMORY AREA DEFINITIONS FOR LINKER

The LINKER needs information about several important memory areas of the operating system before it can begin linking individual modules together. On the page after the module print statements, these memory areas are defined for the LINKER. These are not detailed memory area definitions. They contain enough information to let the LINKER begin operation and no more.

This information is not commonly used during operating system support.

OVERLAY DEFINITIONS

After the initial LINKER memory area definitions, the listing contains several pages that show parameter definitions for specific modules of the operating system.

- **Default Patch Area Size:** At the top of this page is one line that defines the default size of the patch area for a module. All modules have a patch area of this size unless a different patch area size is explicitly defined.
- **Overlay Definitions:** The most important information on these pages shows which parts of the operating system are defined as overlays. All overlay definitions are contained within the segment definition of the global module ("seg global_ mod" in the listing). The names of the overlays are the names included in the Module Interface Definition (MID) of the operating system. These names are related to, but not the same as the three-character mnemonics used in section 1, Operating System Modules. To relate these overlay names to three-character mnemonics, use the Overlay ICM list included later in the LINKER listing.

HEAP MEMORY AREA ALLOCATION (NON-DEFAULT)

The MCP/VS operating system contains several memory environments that are used by specific modules. The sizes of the memory areas within these memory environments are defined by the LINKER using default values. A few of these memory environments require heap memory areas that are larger or smaller than the default size.

One page of the LINKER listing contains definitions for several module-specific memory environments. These definitions explicitly create patch areas and heap memory areas (using sizes other than the defaults). This information is not commonly used during operating system support.

MODULE ICM LIST

After the special memory area definitions, the listing includes a module ICM list. This list is a set of instructions for the LINKER. It tells the LINKER which Independently Compiled Modules (ICMs) to link into the operating system and the location of each ICM on the development center's computer systems.

The ICMs in this list contain modules, not overlays. In this instance, the term "module" is used specifically, rather than the general sense used in the rest of this document. A list of ICMs linked as overlays is included after the module ICM list.

Figure 2-1 shows the first page of a module ICM list.

```
435                                                                    40000000 EDITOR
436      icm                                                           40001000 EDITOR
437   % ------------------------------------------------------------    40002000 EDITOR
438   %|                                                            |   40003000 EDITOR
439   %|                        Module ICM's                        |   40004000 EDITOR
440   %|                                                            |   40005000 EDITOR
441   % ------------------------------------------------------------    40006000 EDITOR
442   %                                                                40007000 EDITOR
443           MIDIno.mcpvs                        ON LISTER,           40008000 EDITOR
444   %                                                                40009000 EDITOR
445   %                 PROCESSOR UNIQUE MODULES                       40010000 EDITOR
446   %                                                                40011000 EDITOR
447           BOOIno.bootstrap                    ON LISTER,           40012000 EDITOR
448           SINIno.system_initialization        ON LISTER,           40013000 EDITOR
449           KERIno.kernel_mod                   ON LISTER,           40014000 EDITOR
450           MMGIno.mem_mgr                      ON LISTER,           40015000 EDITOR
451           IOMIno.io_mod                       ON LISTER,           40016000 EDITOR
452           FAUIno.fault_handler_module         ON LISTER,           40017000 EDITOR
453           FHSIno.faultsupport_module          ON LISTER,           40018000 EDITOR
454           DBGIno.debug_module                 ON LISTER,           40019000 EDITOR
455           TRCIno.trace_module                 ON LISTER,           40020000 EDITOR
456   %                                                                40021000 EDITOR
457   %                 GENERAL SYSTEM SERVICES                        40022000 EDITOR
458   %                                                                40023000 EDITOR
459           GLBIno.global_mod                   ON LISTER,           40024000 EDITOR
460           CFMIno.cf_io                        ON LISTER,           40025000 EDITOR
461           DMSIno.dms2                         ON LISTER,           40026000 EDITOR
462           LIOIno.liom                         ON LISTER,           40027000 EDITOR
463           DIOIno.dirio                        ON LISTER,           40028000 EDITOR
464           UQKIno.user_quickmem_mod            ON LISTER,           40029000 EDITOR
465           SECIno.sys_access_security          ON LISTER,           40030000 EDITOR
466           CALIno.procl                        ON LISTER,           40031000 EDITOR
467           MCRIno.micr                         ON LISTER,           40032000 EDITOR
468   %                                                                40033000 EDITOR
469   %                       STATUS                                   40034000 EDITOR
470   %                                                                40035000 EDITOR
471           STSIno.status                       ON LISTER,           40036000 EDITOR
472           CWTIno.complex_wait                 ON LISTER,           40037000 EDITOR
473           RESIno.resmod                       ON LISTER,           40038000 EDITOR
474           WTGIno.wtgmod                       ON LISTER,           40039000 EDITOR
475           TMRIno.timmod                       ON LISTER,           40040000 EDITOR
476   %                                                                40041000 EDITOR
477   %                     KEYBOARD                                   40042000 EDITOR
478           KBMIno.kbout_mod                    ON LISTER,           40043000 EDITOR
479           OIOIno.ocsio                        ON LISTER,           40044000 EDITOR
480           DRVIno.ocdrv                        ON LISTER,           40045000 EDITOR
481           CNDIno.comnd                        ON LISTER,           40046000 EDITOR
482           RJEIno.rje_mod                      ON LISTER,           40047000 EDITOR
483           KSMIno.sysmaint_commands            ON LISTER,           40048000 EDITOR
484           KWYIno.kb_wy                        ON LISTER,           40049000 EDITOR
485           OCSIno.ocsmod                       ON LISTER,           40050000 EDITOR
```

Figure 2-1. Module ICM List

Each line in the module ICM list is formatted as follows:

```
XXXIuc.yyyyyy_ yyyyyy                    ON PPPPPP
```

- **XXX** represents the three-character mnemonic that identifies the module. This is the same mnemonic used in section 1, Operating System Modules.
- **I** identifies the file as an ICM.
- **uc** represents the user code of the ICM file on the development center's computer systems.
- **yyyyyy_ yyyyyy** represents the extended name of the module as defined in the Module Interface Definition (MID). To associate MID names with three-character mnemonics, use the module ICM list or the overlay ICM list that follows.
- **PPPPPP** represents the family name of a diskpack on the development center's computer systems.

The module named "mcpvs" is the MID of the operating system. It contains definitions of all data structures used in common, entry points and layouts of the procedures within each module.

The comments designated with a percent sign (%) in column 1 divide the modules into categories. These categories are only for informal use in operating system development; they do not affect the operating system in any way.

OVERLAY ICM LIST

The overlay ICM list is an extension of the module ICM list that precedes it. The division of the ICM list into modules and overlays is for documentation purposes only. The LINKER obtains information about modules and overlays from the segmentation statements in the LINKER control language.

The layout of the overlay ICM list is similar to that of the module ICM list. Figure 2-1 shows an example.

INTRINSIC LIST

The intrinsic list is logically similar to the ICM list that precedes it. The intrinsic list tells the LINKER which programs and data files to link into the operating system as bound intrinsics. It also gives the location of each file on the development center's computer systems.

Each line of the intrinsic list is formatted as follows:

```
XXXXXX  AS  YYYYYY   ON PPPPPP
```

- **XXXXXX** represents the actual name of file containing the intrinsic program or data file.
- **YYYYYY** represents the name that the program or data file is given as it is linked into the operating system as a bound file.
- **PPPPPP** represents the family name of a diskpack on the development center's computer systems.

LINK LIST OF MODULES

After the intrinsic list, the LINKER prints a list of all modules (in the general sense) that are linked together to form the operating system. This portion of the LINKER listing is similar to a compile listing or a binder listing for an application program. This list is generated as the operating system is linked together.

Figure 2-2 shows the first page of this list.

```
      MODULE NAME                ICM NAME        COMPILED BY      COMPILE DATE     MID NAME      MID COMP. DATE
============================== ============== ================== ============== ============== ==============

mcpvs                          LISTER/MIDIno SPRITE 1140   87/117 20:40 07/13/87

bootstrap                      LISTER/BOOIno SPRITE 1140   87/117 23:07 07/14/87 mcpvs          20:40 07/13/87
     ERROR:  INCONSISTENT CONTENTS FOR SEGMENT TABLE 2 IN bootstrap (LEVEL = 4)
     ERROR:  INCONSISTENT CONTENTS FOR SEGMENT TABLE 0 IN bootstrap (LEVEL = 4)

system_initialization          LISTER/SINIno SPRITE 1140   87/117 00:43 07/15/87 mcpvs          20:40 07/13/87

kernel_mod                     LISTER/KERIno SPRASM 1140   87/117 23:06 07/14/87 mcpvs          20:40 07/13/87

mem_mgr                        LISTER/MMGIno SPRITE 1140   87/117 01:59 07/15/87 mcpvs          20:40 07/13/87
     ERROR:  INCONSISTENT CONTENTS FOR SEGMENT TABLE 22 IN mem_mgr (LEVEL = 4)
     ERROR:  INCONSISTENT CONTENTS FOR SEGMENT TABLE 2 IN mem_mgr (LEVEL = 4)
     ERROR:  INCONSISTENT CONTENTS FOR SEGMENT TABLE 0 IN mem_mgr (LEVEL = 4)

io_mod                         LISTER/IOMIno SPRASM 1140   87/117 16:58 07/15/87 mcpvs          20:40 07/13/87
     ERROR:  ENTRY LABEL 0 IN io_mod.io_mod NOT DEFINED (LEVEL = 6)

fault_handler_module           LISTER/FAUIno SPRASM 1140   87/117 02:07 07/15/87 mcpvs          20:40 07/13/87

faultsupport_module            LISTER/FHSIno SPRASM 1140   87/117 05:46 07/15/87 mcpvs          20:40 07/13/87

debug_module                   LISTER/DBGIno SPRITE 1140   87/117 06:34 07/15/87 mcpvs          20:40 07/13/87
     ERROR:  INCONSISTENT CONTENTS FOR SEGMENT TABLE 2 IN debug_module (LEVEL = 4)

trace_module                   LISTER/TRCIno SPRITE 1140   87/117 06:04 07/15/87 mcpvs          20:40 07/13/87

global_mod                     LISTER/GLBIno SPRASM 1140   87/117 22:16 07/14/87 mcpvs          20:40 07/13/87

cf_io                          LISTER/CFMIno SPRITE 1140   87/117 23:15 07/14/87 mcpvs          20:40 07/13/87

dms2                           LISTER/DMSIno SPRITE 1140   87/117 03:19 07/15/87 mcpvs          20:40 07/13/87

liom                           LISTER/LIOIno SPRASM 1140   87/117 22:19 07/14/87 mcpvs          20:40 07/13/87

dirio                          LISTER/DIOIno SPRASM 1140   87/117 03:11 07/15/87 mcpvs          20:40 07/13/87

user_quickmem_mod              LISTER/UQKIno SPRASM 1140   87/117 02:08 07/15/87 mcpvs          20:40 07/13/87

sys_access_security            LISTER/SECIno SPRASM 1140   87/117 03:24 07/15/87 mcpvs          20:40 07/13/87

procl                          LISTER/CALIno SPRITE 1140   87/117 23:44 07/14/87 mcpvs          20:40 07/13/87

micr                           LISTER/MCRIno SPRASM 1140   87/117 05:16 07/15/87 mcpvs          20:40 07/13/87

status                         LISTER/STSIno SPRASM 1140   87/117 17:03 07/15/87 mcpvs          20:40 07/13/87

complex_wait                   LISTER/CWTIno SPRASM 1140   87/117 00:23 07/15/87 mcpvs          20:40 07/13/87
```

Figure 2-2. Link List of Modules / Overlays

The following information is provided for each module of the operating system.

- **MODULE NAME**: The name of the module as shown in the Module Interface Definition (MID) used by the LINKER. To associate this name with the three-character mnemonic of the module, use the first three characters after the slash in the column immediately to the right (ICM NAME).
- **ICM NAME**: The name of the file containing the Independently Compiled Module (ICM) that is linked into the operating system. This name includes the family name of a diskpack on the development center's computer systems, followed by a slash, followed by the file name.
- **COMPILED BY**: The name, version and compile date *of the compiler* that compiled the ICM that is linked. The compile date is in Julian format. This is not the compile date of the ICM itself.

- **COMPILE DATE**: The time and date that the module was compiled into an ICM.
- **MID NAME**: The name of the Module Interface Definition (MID) that was used when the module was compiled into an ICM. This is not necessarily the same as the name of the MID used by the LINKER to link the operating system.
- **MID COMP. DATE**: The compile time and date of the MID that was used when the module was compiled into an ICM. This is usually, but not necessarily, the same MID as the one used when the LINKER links the operating system together.

The LINKER notes any inconsistencies in the interfaces between modules with a message marked ERROR. The LINKER listing of an operating system that is released for distribution may contain a small number of inconsistencies. These inconsistencies do not affect how the operating system runs. Inconsistencies that affect the operating system in any significant way are resolved before the operating system is released.

SEGMENT CONTENTS

Following the link list of modules, the LINKER prints an outline of the contents of each module or data segment within the operating system. Each outline shows the relative addresses of the procedures within a module or data blocks within a data segment. The segment number of the module or data segment is printed at the top of the outline.

On most occasions when the operating system fails, an environment number and a relative memory address are displayed on the ODT. Using the environment layout (see later in this section) and the segment contents, you can get a quick idea of the module and the procedure within the module where the system failed. A procedure for this is included later in this section.

The segment contents takes up the largest part of the LINKER listing. Figure 2-3 shows one page of the segment contents.

```
CONTENTS OF SEGMENT # 22: ccdsk

  BASE    SIZE              BLOCK NAME
====== ======   ==============================
 66000   2000   ccdsk.ccdsk.CCDSK
 68000  16418          ccdsk
 68000         +        cc_disk_card
 71916         +        cc_moredisk_card
 72886         +        cc_pack_card
 74948         +        add_a_disk
 75486         +        TESTDK
 77546         +        ADDMST
 78648         +        BLDLBL
 79654         +        BLDAVL
 84420    320   ccdsk_ENVPS
 84740   1260   ccdsk.PATCH_AREA


CONTENTS OF SEGMENT # 23: kbdin

  BASE    SIZE              BLOCK NAME
====== ======   ==============================
 66000   5304   kbdin.kbdin.KBDIN
 71304  19864          kbdin
 71304         +        kbdcrd
 71316         +        kbdinc
 75920         +        *WIPE
 81644         +        FUNNY
 91168    280   kbdin_ENVPS
 91448   1552   kbdin.PATCH_AREA


CONTENTS OF SEGMENT # 24: kb_dq

  BASE    SIZE              BLOCK NAME
====== ======   ==============================
 66000    178   kb_dq.kb_dq.KB_DQ
 66178   3512          kb_dq
 66178         +        kbd_dq
 69690   1310   kb_dq.PATCH_AREA


CONTENTS OF SEGMENT # 25: ctlcd

  BASE    SIZE              BLOCK NAME
====== ======   ==============================
 66000   2924   ctlcd.ctlcd.CTLCD
 68924   6494          ctlcd
 68924         +        cc_evl
 73862         +        cmpsch
```

Figure 2-3. Segment Contents

Each outline in the segment contents includes:

- **BASE**: The relative memory address within the module or data segment where a procedure or data block begins. Divisions within a procedure or data block are indicated with a base address.
- **SIZE**: The size (in digits) of the procedure or data block. Divisions within a procedure or data block are indicated with a base address. For modules written in SPRITE, the size of each procedure is provided. For modules written in SPRASM, sizes are listed for most of the procedures or data blocks within the segment. If a plus sign is printed for the size, it indicates that the procedure or data block is included in the previously printed size.
- **BLOCK NAME**: The name of the procedure or data block as declared in the module.

SEGMENT LAYOUT

After the segment contents (toward the end of the listing), the LINKER prints a summary of the segment information that is generated as the operating system is linked together. Most of the information in this summary is duplicated elsewhere in the listing.

Within this summary, segments are listed in numeric order as assigned by the LINKER, using the name included in the Module Interface Definition (MID). Figure 2-4 shows the first page of the segment layout.

SEGMENT LAYOUT

NUMBER	NAME	BASE	LIMIT	CODE FILE RECORD	PATCH AREA BASE	PATCH AREA SIZE	HEAP BASE	HEAP SIZE	OVLY AREA SIZE	
1	data_page	0	10000	4	9321	679				
2	user_data	0	0	5						
3	locb_queue_area	0	0	5						
4	mcp_heap	0	0	5						
5	file_array	0	0	5						
6	debug_poke_data	0	0	5						
7	debug_data	0	78000	5	77283	717				
8	dcp_tag_pool	0	0	10						
9	mcs_queue_area	0	0	10						
10	dbp_buffer_ma	0	0	10						
11	mcs_trash_area	0	0	10						
12	shared_subport_ma	0	0	10						
13	interrupt_data_area	0	10000	10						
14	kernel_mod	0	15000	11	14810	190				
15	system_tables	0	105000	12	104433	567				
16	global_mod	0	93000	18	64600	1400			27000	
17	io_mod	0	96000	22	94512	1488				
18	system_locb_area	0	0	28						
19	bootstrap_data_segment	0	16000	28	601	15399				
20	cc_dlp	66000	74000	29	72076	1924				OVERLAY 16
21	ccunt	66000	78000	30	76256	1744				OVERLAY 16
22	ccdsk	66000	86000	31	84740	1260				OVERLAY 16
23	kbdin	66000	93000	33	91448	1552				OVERLAY 16
24	kb_dq	66000	71000	35	69690	1310				OVERLAY 16
25	ctlcd	66000	77000	36	75700	1300				OVERLAY 16
26	cc_01	66000	71000	37	69852	1148				OVERLAY 16
27	cc_02	66000	81000	38	79364	1636				OVERLAY 16
28	boj	66000	76000	39	74260	1740				OVERLAY 16
29	trmct	66000	75000	40	73684	1316				OVERLAY 16
30	term	66000	82000	41	80964	1036				OVERLAY 16
31	bna_support	66000	79000	42	77740	1260				OVERLAY 16
32	kb_wt	66000	74000	43	72612	1388				OVERLAY 16
33	kb_to	66000	78000	44	76640	1360				OVERLAY 16
34	kb_pd	66000	77000	45	75468	1532				OVERLAY 16
35	kb_pb	66000	77000	46	75500	1500				OVERLAY 16
36	kb_ss	66000	74000	47	72312	1688				OVERLAY 16
37	kb_db	66000	76000	48	74152	1848				OVERLAY 16
38	kb_rs	66000	77000	49	75192	1808				OVERLAY 16
39	kb_ok	66000	71000	50	69352	1648				OVERLAY 16
40	kb_dc	66000	75000	51	73120	1880				OVERLAY 16
41	kb_in	66000	74000	52	72936	1064				OVERLAY 16
42	kb_qt	66000	71000	53	69604	1396				OVERLAY 16
43	kb_ds	66000	86000	54	84980	1020				OVERLAY 16
44	kb_dl	66000	81000	56	79568	1432				OVERLAY 16

Figure 2-4. Segment Layout

- **BASE**: Modules are listed with a base of 0. Data segments are also listed with a base of 0. All overlays are listed with the same base (the base of the overlay area). Overlays are further designated by the word OVERLAY in the far right-hand column of the page.
- **LIMIT**: The LINKER lists the ending address of the module, overlay or data segment.
- **CODE FILE RECORD**: The LINKER lists the record within the operating system code file where each module, overlay or data segment of the operating system begins.
- **PATCH AREA BASE-PATCH AREA SIZE**: The relative address within each module, overlay or data segment where the patch area begins. The LINKER also includes the size of the patch area. No information is printed for data segments that do not have patch areas.

Other information in the segment layout is not commonly used during operating system support.

ENVIRONMENT LAYOUT

After the segment layout, the LINKER prints an environment layout that contains a small amount of information that is very useful for operating system support. Figure 2-5 shows the first page of the environment layout.

```
ENVIRONMENT LAYOUT
=========== ======

            NAME                TYP NUMB MAT @   #S FLGS  SEGS *=UNUSED,#=ORIG,#.#=COPY USR,#:#=COPY SYS
================================ === ==== ====== == ====  ================================================
user_services_mat                USR    0      0 14 0000  1    2    81   *    3    4    5    6    7    8
                                                          9    10   11   12
kernel_mod                       SYS    0    280  8 0000  13   14   *    *    *    *    *    15
global_mod                       SYS    1    440  8 0000  0.0  16   0.1  *    0.4  0.5  0.6  0:7
io_mod                           SYS    2    600  8 0000  0.0  17   0.1  18   0.4  0.5  0.6  0:7
support_candidates               SYS    3    760  8 0000  0.0  138  0.1  17:3 86   0.5  0.6  0:7
timmod                           SYS    4    920  8 0000  0.0  115  *    *    *    92   *    0:7
message_mod_mat                  SYS    5   1080  8 0000  0.0  *    *    *    *    0.5  78   0:7
user_quickmem_mod                SYS    6   1240  8 0000  0.0  107  0.1  *    *    93   *    0:7
track_mod                        SYS    7   1400  8 0000  0.0  144  0.1  87   *    *    *    0:7
stoq_mod                         SYS    8   1560  8 0000  0.0  136  0.1  79   80   88   *    0:7
sqash                            SYS    9   1720  8 0000  0.0  126  *    *    89   90   91   0:7
bootstrap                        SYS   10   1880  8 0000  19   95   *    *    *    *    *    0:7
cf_environment                   SYS   11   2040  8 0000  0.0  *    0.1  94   0.4  0.5  0.6  0:7
dcp_environment                  SYS   12   2200  8 0000  0.0  *    0.1  85   0.4  0.9  0.6  0:7
dcp_trash                        SYS   13   2360  8 0000  0.12 *    *    *    *    *    *    *
system_access_security_mat       SYS   14   2520  8 0000  0.0  *    0.1  84   *    *    0.6  0:7
complex_wait_mat                 SYS   15   2680  8 0000  0.0  *    0.1  83   *    *    0.6  0:7
io_mod_mat                       SYS   16   2840  8 0000  0.0  *    0.1  2:3  0.4  0.5  0.6  0:7
port_mod_mat                     SYS   17   3000  8 0000  0.0  *    0.1  82   0.13 0.5  0.6  0:7
debug_dummy_mat                  SYS   18   3160  8 0000  0.0  *    0.7  0.8  *    *    *    *
interrupt_mat                    SYS   19   3320  8 0000  0:0  *    *    *    *    *    *    0:7
mcpvs                            SYS   20   3480  8 0000  0.0  *    0.1  *    0.4  0.5  0.6  0:7
mem_mgr                          SYS   21   3640  8 0000  0.0  98   *    *    *    76   77   0:7
system_initialization            SYS   22   3800  8 0000  0.0  96   97   *    0.4  0:0  0.6  0:7
micr                             SYS   23   3960  8 0000  0.0  110  0.1  *    0.4  0.5  0.6  0:7
maint_log                        SYS   24   4120  8 0000  0.0  139  0.1  140  0.4  0.5  0.6  0:7
syst                             SYS   25   4280  8 0000  0.0  143  0.1  *    0.4  0.5  0.6  0:7
debug_module                     SYS   26   4440  8 0000  0.0  101  0.7  0.8  0.4  0.5  0.6  0:7
faultsupport_module              SYS   27   4600  8 0000  0.0  100  0.1  *    *    *    *    0:7
fault_handler_module             SYS   28   4760  8 0000  0.0  99   *    *    *    *    *    0:7
port_mod                         SYS   29   4920  8 0000  0.0  137  0.1  17:3 0.13 0.5  0.6  0:7
complex_wait                     SYS   30   5080  8 0000  0.0  112  0.1  15:3 *    *    0.6  0:7
dirio                            SYS   31   5240  8 0000  0.0  106  0.1  *    0.4  0.5  0.6  0:7
core_to_core_mod                 SYS   32   5400  8 0000  0.0  135  0.1  *    0.4  0.5  0.6  0:7
dcom                             SYS   33   5560  8 0000  0.0  131  0.1  *    0.4  0.5  0.6  0:7
dfsec                            SYS   34   5720  8 0000  0.0  130  0.1  *    0.4  0.5  0.6  0:7
dms2                             SYS   35   5880  8 0000  0.0  104  0.1  0.11 0.4  0.5  0.6  0:7
trace_module                     SYS   36   6040  8 0000  0.0  102  0.1  0.7  0.4  0.5  0.6  0:7
cf_io                            SYS   37   6200  8 0000  0.0  103  0.1  11:3 0.4  0.5  0.6  0:7
iiom                             SYS   38   6360  8 0000  0.0  105  0.1  *    0.4  0.5  0.6  0:7
sys_access_security              SYS   39   6520  8 0000  0.0  108  0.1  14:3 *    *    0.6  0:7
procl                            SYS   40   6680  8 0000  0.0  109  0.1  *    0.4  0.5  0.6  0:7
status                           SYS   41   6840  8 0000  0.0  111  0.1  *    0.4  0.5  0.6  0:7
resmod                           SYS   42   7000  8 0000  0.0  113  0.1  *    0.4  0.5  0.6  0:7
wtgmod                           SYS   43   7160  8 0000  0.0  114  0.1  *    0.4  0.5  0.6  0:7
```

Figure 2-5. Environment Layout

Important information in the environment layout includes:

- **NAME**: The name of the module, overlay or data segment. This name is obtained from the Module Interface Definition (MID).
- **TYP**: The type of module, overlay or data segment. This column is SYS (meaning system) for all environments except the user services memory area table, which is marked USR (meaning user).
- **NUMB**: The environment number of the module, overlay or data segment. This number is the one displayed on the fault screen when the system fails. It is also the number used with the PATCH and DISPLAY commands.

- **SEGS *=UNUSED #=ORIG. #.#=COPY USR #:#=COPY SYS**: The most frequently used information in these columns occurs in the second column from the left (under the * in the header). This number is the segment number assigned to the module, overlay or data segment by the LINKER. It designates the segment contents earlier in the listing.

  As a whole, the numbers in these columns represent the definition of the Memory Area Table for the module, overlay or data segment. The eight columns correspond to the eight entries in the MAT. The way that the number is printed (normal, with a period, with a colon or represented by an asterisk) indicates the type of the MAT entry. The second column from the left corresponds to memory area 1 (the code memory area, and thus represents the segment of operating system code.

Other information in the environment layout is not commonly used during operating system support.

INTRINSIC LAYOUT

Following the environment layout, the LINKER prints an intrinsic layout. This summarizes the information about bound intrinsic files located elsewhere in the listing. The intrinsic layout includes the name of the intrinsic program or data file, the name and location of the file, and other information.

This information is not commonly used during operating system support.

LINKER INFORMATION

The last two pages of a LINKER listing contain statistical information generated as the operating system is linked together. This information is used for comparisons during operating system development.

This information is not commonly used during operating system support.

FINDING THE AVAILABLE PATCH AREA WITHIN A MODULE

This procedure assumes you know which module needs to be patched. Modules are identified by segment number, MID name, three-character mnemonic, or environment number. If you already know the segment number, or MID name proceed to step 2.

1. Obtain a segment number or MID name for the module to be patched.

   a. If you know the three-character mnemonic of the module (example: IOM):

      1) Turn to the module ICM list or the overlay ICM list in the LINKER listing. The module ICM list begins roughly ten pages into the listing and is clearly marked in a comment box. (Figure 2-1 is an example). The overlay ICM list follows the module ICM list.

      2) The first three characters on each line of the module ICM list or overlay ICM list are the three-character mnemonic. Find your three-character mnemonic on the list (example: IOM). Look to the right of the period and obtain the MID name (example io_ mod). Proceed to step 2.

   b. If you know the environment number of the module (example: 2):

      1) Turn to the environment layout of the LINKER listing. The environment layout is near the end of the listing, roughly six pages backward from the last page. (Figure 2-5 is an example).

      2) Find the environment number in the column labelled NUMB. This is the third column from the left side of the page. (Example: 2). Look in the eighth column from the left side of the page. The number in this column is the segment number of the module (example: 17). Alternately, look in the first column on the left side of the page to obtain the MID name (example io_ mod).

      Proceed to step 2.

2. Turn to the segment layout of the LINKER listing. The segment layout is toward to end of the listing, roughly ten pages backward from the last page. (Figure 2-4 is an example)

   Find the desired module using either:

   a. The segment number (example: 17). The segment numbers are listed in the first column from the left side of the page.

   b. The MID name (example: io_ mod). The MID names are in the second column from the left side of the page.

   The sixth and seventh columns from the left side of the page are the base relative address of the patch area and the size of the patch area in digits (examples: 84216 and 1784). For more detailed information about the procedures within the module, use the segment number to look up the module in the segment contents.

NOTE

After finding the patch area of a module, examine the patch area to determine if any patches already exist in the area. If existing patches are present, insert your patch *after* the existing patches. Be careful not to overlap any exisitng patches. If you are changing an existing patch, exercise extreme caution.

TRANSLATING ENVIRONMENT NUMBERS TO SEGMENT NUMBERS (MODULES)

This procedure assumes you have an environment number and want to know what segment number or module of the operating system the environment number represents. Environment numbers are encountered in many situations during operating system support. After a system failure, the fault screen usually shows an environment number and a base-relative address representing the point at which the operating system failed.

To translate an environment number (example: environment number 2):

1. Turn to the environment layout of the LINKER listing. The environment layout is near the end of the listing, roughly six pages backward from the last page. (Figure 2-5 is an example). Segments are listed in order by environment number.
2. Find the environment number in the column labelled NUMB. This is the third column from the left side of the page. (Example: 2).

   a. Look in the eighth column from the left side of the page. This is the segment number of the module (example: 17). Be careful not to confuse the segment number with the number immediately to its left. Segment numbers do not have decimal points.

      After you know the segment number, you can look up the module in the segment layouts of the LINKER listing. The segment layouts show a name, a base-relative address and a size for all of the procedures within a module. If you have a base-relative address from the fault screen, compare it with the base-relative addresses in the segment layout. This tells you the procedure within the module where the failure occured.

   b. Look in the first column on the left side of the page. This is the MID name of the module (example io_ mod). The MID name is the extended name assigned to the module in the Module Interface Definition (MID) of the operating system.

      To look up this module in section 1, Operating System Modules, use the module ICM list in the LINKER listing to find the three-character mnemonic based on the MID name. (Figure 2-1 is an example of the module ICM list.)

SECTION 3
SYSTEM MEMORY DUMPS

A memory dump transfers the contents of a system's memory to some type of storage media. Think of a memory dump as a picture of the operating system's memory taken at a given point in time. Sometimes the phrase memory dump refers to the entity created on the storage media during the transfer (more properly called the memory dump file).

Under the MCP/VS operating system, there are two types of memory dumps: system memory dumps and user program dumps. System memory dumps transfer the operating system's memory, while user program dumps transfer a user program's memory. Memory dump files of the two types of dumps use different formats and require two different dump analysis programs. This section deals with system memory dumps. Information about user program dumps in included in the *V Series MCP/VS Programming Reference Manual*.

This section contains the following material concerning system memory dumps:

- How and when system memory dumps are produced
- Dump printing (DMPANL) options and syntax
- Field descriptions for the DMPANL listings of several frequently used operating system data structures.

HOW AND WHEN SYSTEM MEMORY DUMPS ARE PRODUCED

A system memory dump copies the contents of the operating system's memory into a memory dump file. The memory dump file of the operating system is always named $\$p0001$, where p is the processor number. The dump file is located on either disk, diskpack or on tape, depending on the USE DUMP record in the system configuration file. Dump procedures vary slightly depending on the location of the memory dump file. After a system memory dump is performed, the DMPANL utility can analyze the memory dump file and produce many different types of formatted output.

Under MCP/VS, system memory dumps can be:

- requested by the operator after the operating system fails
- performed automatically after the operating system fails
- performed while the operating system is running (using a specific command)

System Memory Dump Caused by Operating System Failure

Operating system failures can be divided into two types: failures when the operating system *cannot* execute its normal fault-handling procedure and failures when it *can*. The following paragraphs describe how system memory dumps are performed for both types of operating system failure.

Red-Light Faults

Some failures cause the operating system to stop immediately. In these situations the operating system does not trap properly or execute fault-handling code. The Fault Screen is not displayed. For historical reasons, this kind of failure is called a **red-light** fault. Red-light faults are usually caused by a problem in the kernel or in the fault handling portion of the operating system. When a red-light occurs, the processor stops and displays the message shown in figure 3-1 on the primary ODT:

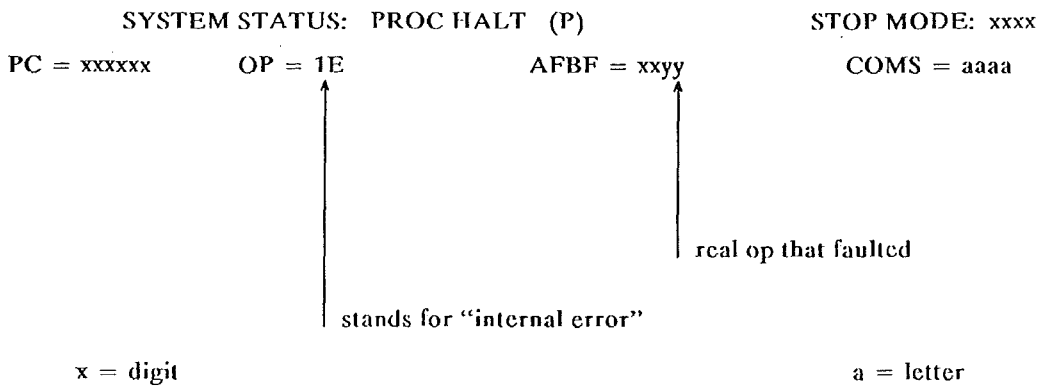


Figure 3-1. ODT Display for Red-light Fault

In this situation, the operator must decide if a memory dump should be performed. If the decision is yes, the DMPMEM utility is used to transfer the contents of system memory to magnetic tape. Information about DMPMEM is provided in the *V Series MCP/VS System Software Operations Guide, Volume 3: System Utilities*.

Operating System Failures (Not Red-light Faults)

Most operating system failures allow the operating system to trap the error and execute fault-handling code properly. When this type of failure occurs, a system memory dump can be performed. The following events take place during an operating system failure that is not a red-light fault.

- The dump-initiation code within the operating system checks the USE DUMP record in the system configuration file. (If the USE DUMP information is corrupt, the results are unpredictable.)
  - If no USE DUMP record exists, or if the USE DUMP record states that memory should be dumped to tape, the system halts. The operator must decide if a memory dump should be performed. The operating system takes no further action.

    If the operator decides to perform a system memory dump, the DMPMEM utility is used to transfer the contents of system memory to magnetic tape. DMPMEM resides on the CV320A flexible disk and is capable of dumping system memory without assistance from the operating system.
  - If the USE DUMP record states that memory should be dumped to disk or diskpack, the operating system begins to perform a system memory dump automatically. As a first step in this process, the operating system checks the integrity of the code that actually writes a memory dump to a storage medium. This dump-writing code is designed to be as compact and error-free as possible. It uses the minimum amount of system resources so that a system memory dump can be written even when parts of the operating system are corrupt.

    The dump-initiation code within the operating system checks the integrity of the dump-writing code using checksums. The entire portion of dump-writing code, including the disk or diskpack address for the dump, is checksummed. This sum is compared against the expected checksum.

    Sums Agree: The dump-writing code transfers the contents of the operating system's memory to disk or diskpack.

    Sums Do Not Agree: Dump-writing code is assumed to be corrupt. The system halts, and the operator must decide whether or not to perform a system memory dump using the DMPMEM utility. Information about DMPMEM is provided in the *V Series MCP/VS System Software Operations Guide, Volume 3: System Utilities*.

Figure 3-2 shows the steps in the creation of a system memory dump. The Fault Screen is created by the operating system, and appears on the ODT whenever the system halts. One option on this screen allows stack traceback, which shows the procedural nesting for the current task by displaying previous stack frames.

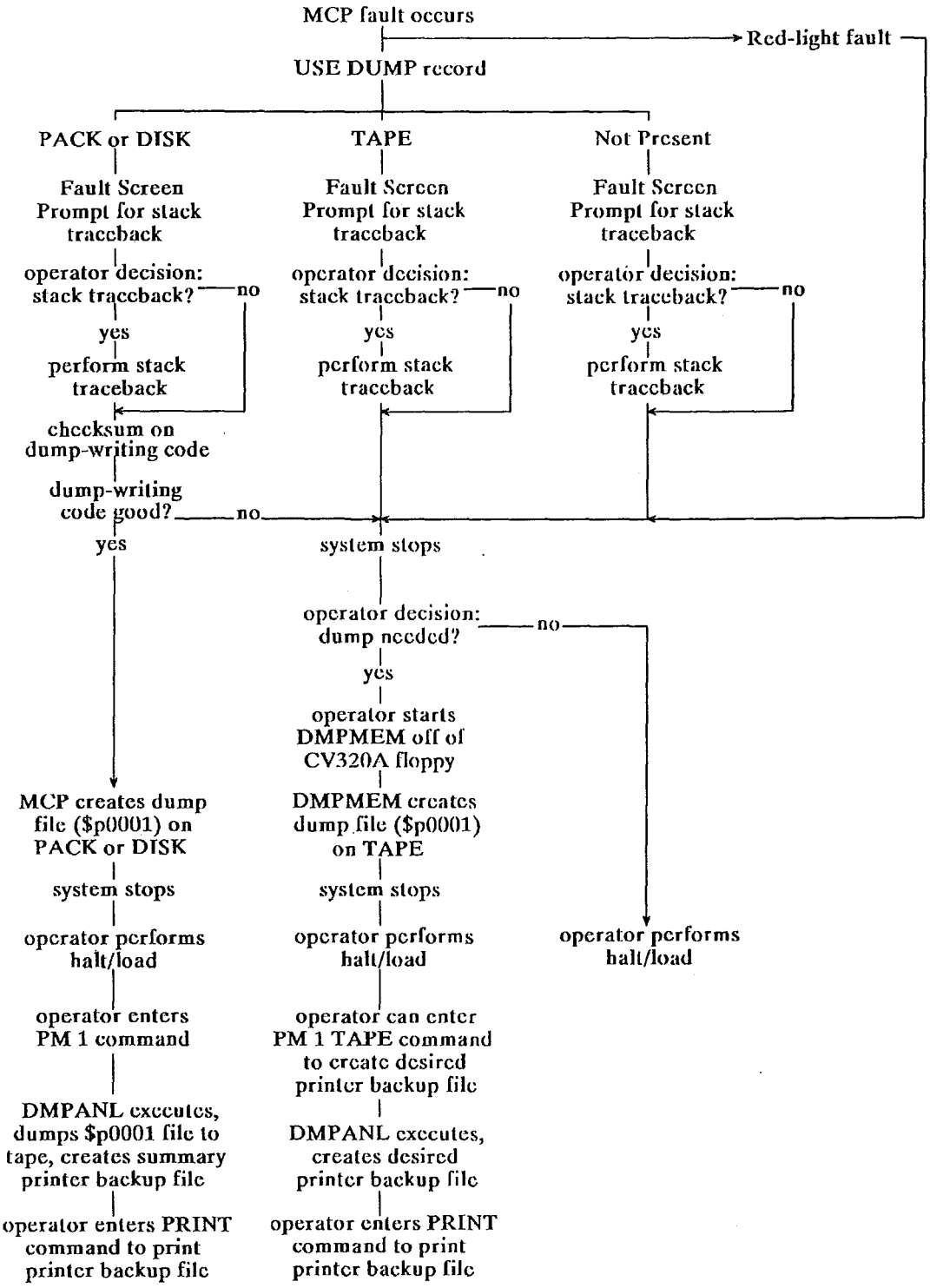


Figure 3-2. System Memory Dump (Operating System Failure)

System Memory Dump Caused by Operator Action

System memory dumps can be created at any time, by entering the DM command, preceded by a mix number of 0. The DM command instructs the operating system to perform a system memory dump and then continue processing.

The function of the 0 DM command depends on the USE DUMP record of the system configuration file.

- 0 DM can only be used if the USE DUMP record directs the memory dump file to disk or diskpack. When 0 DM is entered the operating system begins creating a memory dump file on the indicated medium.
- 0 DM is not allowed if no USE DUMP record is present when the system is cold started. An error message is returned if the command is entered. Without a USE DUMP record, the operating system cannot perform the dump automatically and consequently cannot resume processing. (The only way to perform a dump without a USE DUMP record is to use the DMPMEM utility while the operating system is not running.)
- 0 DM is not allowed if the USE DUMP record directs the memory dump file to magnetic tape. An error message is returned if the command is entered.

The events of this process are shown in figure 3-3.

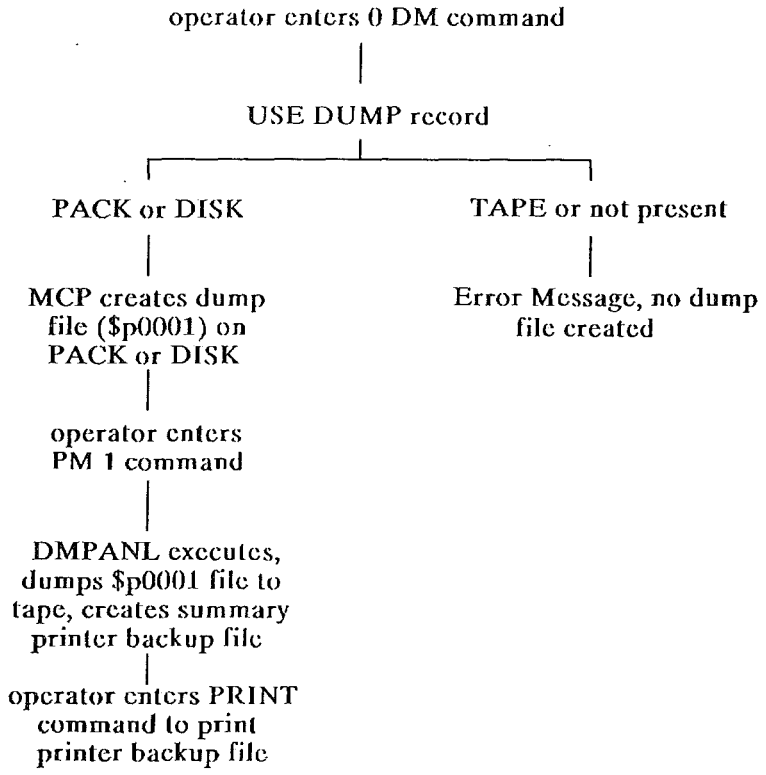


Figure 3-3. System Memory Dump (Operator Action)

PRINTING A SYSTEM MEMORY DUMP (DMPANL)

After a system memory dump is performed, a memory dump file exists on some type of storage medium. The memory dump file of the operating system is always named $\$p0001$, where p is the processor number. It contains the entire contents of the operating system's memory at the point in time when the operating system failed or when the operator entered a 0 DM command.

The intrinsic utility program DMPANL is used to format system memory dumps. DMPANL reads a memory dump file from disk, diskpack, or tape, and produces a printer backup file containing a formatted listing of the dump. The internal name of the printer backup file is MEMORY/DUMP. DMPANL only prints *system* memory dumps, user program dumps are printed by the DMPOUT utility. Information about user program dumps in included in the *V Series MCP/VS Programming Reference Manual.*

DMPANL Requirements

DMPANL changes with each release of the operating system. It is dependent on information about the layout of the memory dump file. This information changes frequently as the operating system is developed and maintained.

To print a system memory dump, the compile date of DMPANL *must be the same* as the compile date of the DMPANL intrinsic bound into the operating system that created the memory dump file. You cannot create a memory dump file on one system and print the dump on another system *unless* the two systems have the same version of DMPANL (with the same compile date). This normally means that the two systems are running the same release of the operating system.

Executing DMPANL (PM1 Command)

To print a system memory dump, use the PM1 command (the 1 refers to the number of the memory dump file). The PM1 command initiates DMPANL. The syntax entered through the PM1 command controls DMPANL. You can enter the DMPANL command syntax in one of two ways:

- As you enter the command PM1, by stringing the commands after the PM1. For example:

  ```
  PM 1 TBL
  ```

- After you enter PM1, by using an asterisk (*) in the command syntax. The asterisk tells DMPANL to wait for commands in the form of accept messages — <mix number>AX <DMPANL syntax>. For example:

  ```
  you enter:              PM 1 *
  the system responds:    ** $00001/DMPANL=05 ACCEPT
  you enter:              5 AX TBL END
  ```

  If you use an asterisk and AX messages to enter DMPANL syntax:

  - You must enter END to end the DMPANL command syntax.
  - You can enter multi-line input.
  - You can enter <mix no>AX HELP to get a list of DMPANL parameters. DMPANL then awaits input.
  - When a syntax error is detected in the parameters, an error message is displayed and all parameters following the one in error are ignored. DMPANL then continues to await input.

Figure 3-4 shows the basic PM1 command syntax.

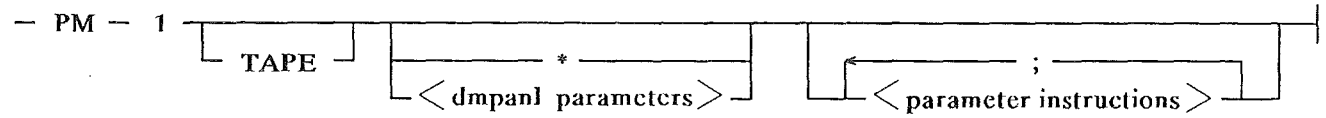


Figure 3-4. PM Command Syntax for System Memory Dumps

null

If you enter the PM1 command with no parameters, the default syntax is used. The default syntax is described later is this section.

TAPE

The source of the memory dump is tape. The tape could have been created with the DMPMEM utility, or by DMPANL (if the default syntax is used). If no other parameters beside TAPE are entered, DMPANL produces a brief report containing general system information and information about the task that was running when the system failed.

*

If you enter an asterisk, DMPANL will issue an ACCEPT and await input. You can enter multi-line input this way.

<dmpanl parameters>

Keywords that control what DMPANL includes in the formatted listing of the system memory dump. DMPANL keywords are explained under DMPANL Parameter Syntax on the following pages.

<parameter instructions>

MCP control instructions that condition the execution of DMPANL, (including FILE, AFTER, CHARGE, MEMORY, LOCK, and so forth). For example:

```
PM 1 TBL ; FILE MEMORY/DUMP = DMPFIL MTP
```

DMPANL Default Syntax

If you enter the PM1 command with no parameters, the results depend on the USE DUMP record in the system configuration file.

- If the USE DUMP record specifies disk or diskpack, DMPANL produces a brief summary listing and copies the entire memory dump file to a magnetic tape.
- If the USE DUMP record specifies TAPE, DMPANL produces an error message. The keyword TAPE must be entered to read a memory dump file from magnetic tape.

For a discussion of the USE DUMP record, see the *V Series System Software Operation Guide, Volume 1: Installation.*

DMPANL Parameter Syntax (PM1 Command)

The internal tables of the MCP/VS operating system take up a reasonably large amount of memory. Because of this, it is practical and important to limit the information that is placed in the formatted listing of a system memory dump.

To control what is included in the formatted printer backup file, use the keywords of the DMPANL syntax to choose dump-printing options. DMPANL keywords let you control what is included in the formatted listing, down to the level of specific tables. The order of most DMPANL keywords is not important; if a given table is requested anywhere in the PM1 commands it is printed.

The keywords are shown in figure 3-5 and described in the paragraphs following the diagram. The default syntax of the PM1 command is detailed earlier in this section.

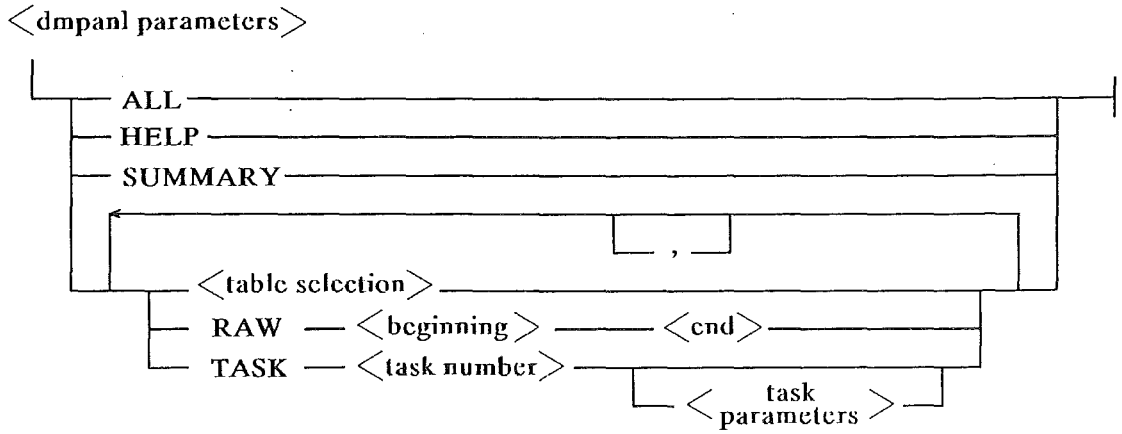


Figure 3-5. DMPANL Parameters

ALL

This option prints all of the loaded MCP, the system tables, and all other allocated memory.

In most cases, the listing produced by the ALL option will exceed the maximum disk or diskpack printer backup file size. If this option is used, the formatted listing should be directed to tape using a file equate statement.

HELP

This option produces a list of the DMPANL parameters. The HELP parameter is normally entered after an asterisk (*). In this case, DMPANL continues to await input after producing the parameter list. If HELP is entered directly after the PM1 command, the parameter list is displayed on the ODT and DMPANL is executed using the default syntax.

SUMMARY

This option prints a brief summary of the system memory dump, including the current task information, the function path, and the current task MCP stack frame.

<table selection>

Keywords that control which tables DMPANL includes in the formatted listing of the system memory dump. Table selection parameters are described later in this section. Their syntax is shown in Figure 3-6 and each parameter is described on the pages following the figure.

RAW <beginning><end>

This option prints a raw (unformatted) listing of memory from the <beginning> memory address to the <end> memory address. Memory addresses can range from the operating system base (entered as 0) to the end of memory on the system. The operating system base is absolute memory address 10,000. No system tables are printed.

TASK <task number>

This option prints selected information relating to an individual task on the system. <task number> is the mix number of an individual task that was running when the operating system failed or when the system memory dump was performed.

<task parameters>

Keywords that control the selection of information about an individual task. Task keywords are described under Task Parameters later in this section. Their syntax is shown in Figure 3-7 and each parameter is explained on the pages following the figure.

Table Selection Parameters

Enter table selection parameters as part of a PM1 command to control what internal data structures (tables) are included in a system memory dump. Figure 3-6 shows the syntax for these parameters.

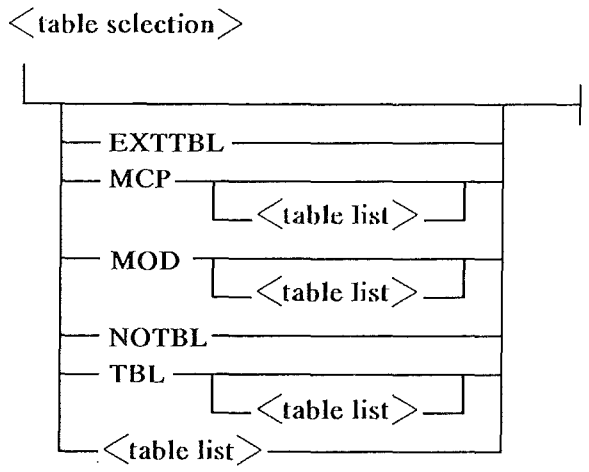


Figure 3-6. Table Selection Parameters (DMPANL)

EXTTBL

This option prints the system tables (see TBL option in the following paragraphs) *and* additional operating system tables. No raw memory dump is produced. The additional tables include the following:

- Halt/Load Parameters
- Mix Table
- EU Table and Disk Subsystem Table
- File Information Blocks (FIB)
- External FIBs (maintained by operating system)
- Disk File Headers (as stored in memory)
- Address Blocks
- Device Alternate Buffers
- Port Global Data Memory Area and Candidate List Memory Area
- Channel Table, Queue R/D Area, I/O Queue Elements
- Memory Area Tables
- Memory Area Status Table
- Segment Dictionary
- Input/Output Assignment Table (IOAT), also known as Device Assignment Table

MCP

Produces a system table breakout and a raw dump of the MCP and overlay area.

In most cases, the listing produced by the MCP option will exceed the maximum disk or diskpack printer backup file size. If this option is used, the formatted listing should be directed to tape using a file equate statement.

MOD

This option prints the system tables, the memory from 0 KD to the top of the overlay area, any MCP modules that were loaded when the dump was performed, and any Independent Runners that were executing when the dump was performed.

In most cases, the listing produced by the MOD option will exceed the maximum disk or diskpack printer backup file size. If this option is used, the formatted listing should be directed to tape using a file equate statement.

NOTBL

This option overrides the DMPANL default option TBL. When NOTBL is entered, DMPANL prints *only* the information specifically requested through the PM1 command. If NOTBL is not entered, DMPANL defaults to the TBL option and prints the system tables along with the requested information.

TBL

This option prints the system tables without a raw memory dump. This is a default DMPANL option. The system tables consist of the following information:

- Interrupt Data Area
- Reserved for Memory Error Report
- Kernel Pointer
- Kernel Request Area
- SYSRED Information
- Channel Descriptors
- Pointer Reference Table
- Port BNA Global Information
- Complex Wait Counters
- DCP Memory Area Table Information
- Disk Directory Global Variables
- Disk File Security Environment Pointer
- DMSII Information
- ISC Information
- Loader Information
- Logical MCP Port Information
- Logical MCP Port Flush Information
- System Lock Structures
- Miscellaneous MCP Variables and Structures
- Mix Related Data
- IC Memory Parity Error Table
- Status Information (NSEC variables)
- ODT Global Information
- Global Scan Word Information
- PCR Work Area
- QWIK Information
- Run Log Information
- Shared Systems Information
- ODT Pointers and Flags
- TRAK Information
- Security Disk Addresses
- System Access Security Information
- Global Waiting Condition Variables
- DCP Pointer Reference Table

- DCP Lock Area
- Last 100 Segments Called List
- Block Lockout Table Information

<table list>

A list of keywords that refer to a specific operating system table or a group of operating system tables. If more than one keyword is entered, the keywords are separated by blank spaces or commas. DMPANL keywords are described on the following pages.

Table Selection Keywords

The following paragraphs describe the formatted listing that DMPANL generates for each DMPANL keyword. DMPANL keywords are entered through the PM1 command and can be included in a list separated by spaces or commas.

CMPLX

This option prints information about tasks that have issued complex wait BCTs and are waiting at the time the system memory dump is performed. The information is from the complex wait table (an internal table of the CWT module). Each entry in the DMPANL listing represents a 42-digit entry in the complex wait table. Each table entry represents one condition in a complex wait BCT issued by an individual task.

The complex wait table consists of eleven linked lists. There is one list of available entries and ten lists of linked entries; one for each type of wait condition (see wait type in the following paragraphs).

The printed information includes:

- The header of each entry gives the memory location of the complex wait table entry. The environment and memory area of this location are the same for all entries.
- The common fields in all entries include:

  CWTNXT: A pointer to the next entry in the linked list for this wait type.

  CWTMIX: A pointer to the mix table entry for the task that is waiting on this condition. (One task can wait on many different conditions).

  CWTTSK: The task number of the task that is waiting on this condition. (One task can wait on many different conditions).

  CWTUSG: The "using" number of this wait condition. The "using" number is defined by the user program in the complex wait BCT.

  CWTTYP: A numeric code representing the type of wait condition. Values include:

  0000 = Doze
  0001 = ODT
  0002 = Output (used by PORTs)
  0003 = Input (used by PORTs, DCP, ISC, RJE, SPO Tank)
  0004 = Change (used by PORTs, DCP, ISC)
  0005 = Ready (used by PORTs)
  0006 = STOQ Input
  0007 = STOQ Output
  0008 = CRCR Input
  0009 = CRCR Output
  0010 = MCP Message

- Other fields of the complex wait table entry are used differently depending on the type of the wait condition.

DCP

This option prints information related to Data Communication Processors (DCPs) and Message Control Systems (MCSs). Printed information includes the DCP Station Table, the MCS Table and the DCP Table.

- **DCP Station Table**: Lists all stations declared on the system. Stations are listed in order using the host-relative logical station number (ST-LSN). The printed information for each station includes:

  Indexes into MCS Table. Index for the station's initialized MCS and current MCS (ST-ID# and ST-MCS).

  Pointer (memory area and offset) to relevant entry in DCP Table (ST-DCO).

  Logical DCP number of DCP that the station is assigned to (ST-DC#). Logical DCP number is associated with the channel number on the DCP's UNIT record.

  Mix number of MCS that the station is assigned to (ST-MIX).

  Physical station number of the station (ST-PSN). The physical station number is obtained from the MCPNpF file and is DCP-relative.

  Station attach flag (ST-ATT)

  Output queue limit of the station (ST-QOT). The number of datacomm write requests that can be accepted by the station. When this is zero, the station is marked inhibited.

  Station inhibited flag (ST-INH)

- **MCS Table**: Lists all Message Control Systems (MCSs) declared on the system. The printed information for each MCS includes:

  Name of MCS file (SX-MCS)

  Mix number of MCS (SX-MIX)

  Logical ID number of MCS (SX-ID#)

  Run log number of MCS (SX-RL#)

  Environment-relative pointer to MCS buffer (SX-BUF)

  Size of MCS buffer (SX-SIZ)

  MCS status flag (SX-STF)

  Max. number of messages MCS buffer can hold (SX-TGQ)

  LSN of last station MCS communicated with (SX-LSN)

  Pointer to attached DCP list (SX-DCP)

  Attached DCP list. List of DCP that the MCS communicates with. Includes link to next entry in list (AT-FWD) and pointer to associated entry in DCP Table (AT-DCP).

  MCS Queue. Appears for MCSs that use queueing. Contains links between queue entries, size of entries, maximum number of entries, and so forth.

  MCS Buffer. Appears only if MCS has opened a DCP file.

- **DCP Table**: Lists the Data Communications Processors (DCPs) declared on the system. The printed information for each DCP includes:

Pointer to default read buffer for the DCP (OH-DFR)

Size of default read buffer (OH-SIZ)

Pointer to IOAT for the DCP (OH-IOA)

Logical, host-relative number of the DCP (OH-DC#)

Status digit for each MCS communicating with the DCP (OH-STA). Relates to SX-STF in MCS Table.

Type of the DCP (OH-TYP). Relates to hardware subtype in IOAT (IO-HDS).

Task number of independent runner that is reading from the DCP

Default read (DFR) buffer. Contains information from the last read from the DCP prior to system memory dump.

Device Assignment Table entry for the DCP. Also known as the "hard IOAT".

DMS

This option prints the information in two internal DMSII data structures; the Database Program (DBP) Table and the Database User Program Control Table.

- **Database Program (DBP) Table**: contains one entry for each DBP or ISAM handler program on the system.
- **Database User Program Control Table**: contains one entry for each user program that communicates with a DBP or ISAM handler, including one entry for the DBP or ISAM handler itself.

EU

This option prints the information in the EU Table (which contains ID numbers) and in the Disk Subsystem Table.

- **EU Table**: Contains one entry for each disk or diskpack unit (disk ID) on the system. Some of the information in an EU Table entry is summarized in the following list.

  Name of the disk (EU-FAM)

  Hardware type (EU-HDW) and drive type (EU-HDS)

  DLP type (EU-TYP)

  Channel and unit of drive (EU-CHN)

  EU Table index (EU-EU#). *Not* the same as disk ID number.

  Diskpack serial number (EU-PSN)

  Logical disk subsystem number (EU-SB#)

  Device number as declared, FLAME index (EU-DV#)

  Number of MCP accesses to EU table entry (EU-MOC)

  Mix number of task using device via direct I/O (EU-TSK)

  Disk ID number in hexadecimal (EU-ID)

Total error count for EU (EU-ERC)

Number of allowed retries (EU-ERT)

Block count for EU (EU-BCT)

Cumulative I/O time during status period (EU-IOT)

Device status flags for EU, including binary/decimal addressing flag (EU-BIN).

Pointers to next device in linked lists. Lists include subsystem (EU-SBP), default subsystem (EU-DEP), shared subsystem (EU-DSP), all diskpacks (EU-DPK), and all disks (EU-DSK).

Pointer to IOAT entry for SSP (EU-SSP)

Record number of device in configuration file (EU-CRD)

Average I/O utilization (EU-AV)

Number of disk areas or number of diskpack files open (EU-FOC)

EU Table lock structure. Not currently used. (EU-LOK)

Lowest unprotected disk address (EU-LOW)

Total availabledisk space, in sectors (EU-TAS)

Disk address of device's available table entry (EU-FAS)

Disk address of device's directory entry (EU-FDS)

Virtual result descriptor of last status I/O. Not currently used. (EU-VRD)

Highest disk sector address. This is primarily used for SMD devices (EU-HAD).

Sectors per cylinder (EU-SPC)

- **Disk Subsystem Table**: Shows the order of disk and diskpack devices in the device lists maintained by the operating system. For each list, the listing shows the list name, the subsystem status flag (DS-FLG), a pointer to the EU table entry of the first device on the list (DS-LNP) and pointers to the EU Table entries of the remaining devices in the list. The operating system maintains the following device lists:

  Subsystem (1 list per declared disk subsystem)

  Default disk

  Shared disk

  All disk

  Family (1 list per declared diskpack family)

  Unrestricted diskpack

  All diskpack

FILE

This option prints specific information about every open file on the system. Printed information includes:

- File Information Zone (FIZ)
- External FIB (maintained by operating system)
- Device Assignment Table — also known as Input Output Assignment Table (IOAT)
- File Header (as stored in memory)
- Address blocks
- I/O Queue Elements
- File Information Block (FIB)
- Device alterate buffer

For open PORT files, printed information includes the port attribute structure, the subport directory, and the subport attribute structure.

GLOBAL

This option prints a raw memory dump of the operating system module knwn as global (GLB). This module includes global operating system code and the overlay segment.

HLPARMS

This option prints the halt load parameters, as they are stored by the operating system at the time of the system memory dump. Definitions of halt/load parameters are included in the Module Interface Description (MID) of the operating system. The following list summarizes some of the halt/load information printed by DMPANL:

- Halt/load format code, date and time of the halt/load parameter information, next run log entry ID number.
- Absolute disk addresses of operating system files. Includes program name table, master available table, working available table, PCR directory, disk directory, H/L parameters and others.
- System log file indexes and limits; number of next run, maintenance, and ODT log files.
- Values obtained from LIMIT and CONTROL records of the system configuration file. Includes maximum number of jobs in mix, limits on DMS operation, limits on STOQUE operation, priorities for various categories of jobs (DMS, RJE, WFL), and others.
- Module option bit flags (initially obtained from USE records of the system configuration file). Options include MICR, DCOM, STOQ, DMS2, CMPX, RLOG and many others. Bit values interpreted as follows:

  1 bit = Option is set
  2 bit = Module is loaded
  4 bit = Module is in use

- System option bit flags (initially obtained from USE records). Options include logging, library maintenance, printer, security, job termination, system memory dump and others. Bit values for logging options are interpreted as follows:

1 bit = Option is set
2 bit = AUTO
4 bit = Dependent on option
8 bit = Dependent on option

Other system options represent four options per digit. Detailed information is available in the Module Interface Description (MID) of the operating system.

- Lowest backup file number present and last backup file number used for printer, punch, dump and card backup files.

IOAT

This option prints the information in the Input/Output Assignment Table (IOAT). This data structure is also known as the Device Assignment Table. It contains "hard" IOATs for standard devices and device files. The DMPANL listing of the IOAT shows fields that are common to all IOAT entries and additional fields that are dependent on the type of device.

A detailed listing of the IOAT entry is contained in the Module Interface Description (MID) of the operating system. Several important portions of common IOAT information are summarized in the following list.

- Hardware type and supplementary hardware type of the device (IO-HDP, IO-HDS)
- Channel and unit numbers of the device (IO-CHN, IO-UNT)
- Number of queued I/Os for this device (IO-QUE)
- I/O statistics counts, including total error count (IO-ERC), number of retires on failed I/Os (IO-ERT), and block count (IO-BCT)
- File ID and multifile ID (IO-ID, IO-MFD)
- Name of firmware file loaded to this device (IO-MFD)
- Waiting bit flags (IO-WIO). Includes waiting close flush queue, queued status ops, waiting position or file flush, and waiting I/O complete flags.
- Mix number of job owning device; used with direct I/O. (IO-TSK)

IOQUE

The option prints the Channel Table, the Queue R/D area, the in-process I/O queue elements and the waiting I/O queue elements.

- The Channel Table contains one entry for each channel declared on the system. The entries are printed in order by channel number. Each entry contains information about the specific channel. Some of the important channel information includes:

  Channel number
  Hardware type
  Exchange number
  Firmware level, file ID and family
  Controller information
  Result descriptor (R/D)
  Priority (and priority link)

- The Queue R/D Area contains one entry for each channel declared on the system. The entries are printed in order based on channel priority. Some of the important Queue R/D information includes:

Channel number
Exchange number
Channel flags (including channel inhibited and channel busy)
Offset of in process I/O
IOT result descriptor
DLP result descriptor

- The in process and waiting I/O queue elements are printed in full.

  The I/O queue elements are often used for operating system support. Because of their importance, the individual fields are briefly defined under Commonly Used Data Structures later in this section. More detailed definitions of the fields are provided in the Module Interface Description (MID) of the operating system.

MAST

This option prints the Memory Area Status Table (MAST) and prints where memory is available on the system. The MAST contains one entry for each memory area on the system. Some of the MAST information is summarized in the following list:

- Raw MAST entry. The 40 digits of the actual MAST entry as stored by the operating system.
- Lock digits (2). Used internally by processor.
- Memory area status. Internal status flags.
- Waiting digit. Includes I/O inhibited memory area, memory area to be rolled out and memory area present flags.
- I/O count. Number of I/Os in process to the memory area.
- MA owner. Mix number of task that currently owns the memory area.
- ET number. Environment number of original.
- MAT number. Memory area number of original.
- Duplicate in task. Only used if "duplicate original" exists for use of TRC-IR independent runner.
- Link to next avilable entry.

MAT

This option prints the Memory Area Table (MAT) entries of every task and independent runner on the system. The printed information includes:

- Raw MAT entry. The 20 digits of the actual MAT entry as stored by the operating system.
- Task number. The mix number of the task or independent runner that this MAT entry belongs to.
- Environment number. An index into the Environment Table (ET) of the task. This number defines which MAT this MAT entry is a part of. (One task can have many MATs).
- MAT status. The type of this MAT entry. There are four MAT entry types:

  Copy
  Original
  Unused
  Faulted

  For more information about the different types of MAT entries, see the *V Series MCP/VS Architecture Manual.*
- Base. Starting memory address of memory area. Printed only for original-type MAT entries. Hardware-dependent absolute memory address in KD.
- Limit. Ending memory address of memory area. Printed only for original-type MAT entries. Hardware-dependent absolute memory address in KD.
- Delta. Amount of "extra" memory allocated to this memory area. In this context, "extra" memory refers to memory that the memory manager module allocates to this memory area in excess of the memory requests received by the memory manager. Printed only for original-type MAT entries.
- MAST index. An index into the Memory Area Status Table (MAST) that is maintained by the operating system. The MAST contains one entry for Printed only for original-type MAT entries.
- MA corruption. A boolean flag indicating memory parity errors during the roll out of this memory to disk. Printed only for faulted-type MAT entries.
- Fault index. An index into an internal table maintained by the memory manager module. Printed only for faulted-type MAT entries.
- ET number. Digits 2 through 7 of the raw MAT entry. Printed only for copy-type MAT entries.
- MA number. Digits 8 and 9 of the raw MAT entry. Printed only for copy-type MAT entries.
- Copy address. A hardware-dependent absolute memory address. Printed only for copy-type MAT entries of type "E".

MIX

This option prints all entries that are in the mix table when the system memory dump is performed. Each entry in the mix table represents one task on the system. The entries are listed in order by task number. The printed information includes:

- The memory address of the mix table entry itself, along with the task number, task name and task status.
- The individual fields of the mix table entry, arranged in order as they are declared in the Module Interface Description (MID) of the operating system.

The mix table entry is often used for operating system support. Because of the mix table's importance, the individual fields are briefly defined under Commonly Used Data Structures later in this section. More detailed definitions of the fields are provided in the MID of the operating system.

PORTS

This option prints the information in the Port Global Data Memory Area and the Candidate List memory area.

- The Port Global Data Memory Area contains an entry for every active PORT on the system. Each entry contains attributes which are shared by all the Subports connected to this PORT, including an event directory.
- The Candidate List memory area contains an entry for every subport that is trying to open. An entry contains attributes necessary for matching subports and control state and status information.

SEC

This option prints all of the entries in the Security User Table. DMPANL prints the following information for each entry.

- Entry type (UST-TP). Possible values are available entry, IOAT entry, PCR entry, mix entry, and linked entry.
- Reference number (UST-RF). Indicates user of this entry (for example MIX-NO or IO-STA).
- User file ID (UST-KY). Identifies access code in USERFL file for this entry.
- User comm link (UST-UT). Pointer to appropriate entry in user combination file.
- Next entry (UST-LK). Pointer to next entry. Used when there is more than one entry for a device (for example LI, BEGINUSER).
- Last entry / previous entry (UST-LS / UST-PV). Pointer from first entry in list to the last entry. For all entries except the first entry this is a pointer to the previous entry (backlink pointer).
- Login type (UST-TY). Possible values include login/logout, BEGINUSER, or USER.
- Security capabilities (UST-ST). Library maintenance capability flag.

SEG

This option prints the segment dictionary of the operating system. The listing contains information about each of the operating system overlays. The printed information includes:

- Segment number of the overlay (S-SEG#). The segment number is a hexidecimal number.
- Number of times the overlay is called (S-CNT)
- Number of times the overlay is located in QWKMEM (S-QCNT)
- Overlay status bit flag (S-STAT). Includes overlay inhibited; no priority; read and store; and read - no store flags.
- Overlay identification flag (S-OVLY)
- Beginning disk address of overlay (S-DISK)
- Size of overlay segment in digits (S-SIZE)

SLOG

This option prints the last hundred records of the ODT log. This information is included in the system tables, but is also printed explicitly through this option. An ODT record includes the following information:

- Record type code (SPO$TP)
- Mix number of associated job (SPO$JN)
- Run log number of associated job (SPO$RN)
- Date record was created (SPO$DT)
- Time of record was created (SPO$TM)
- Record subtype code (SPO$ST) — Input/output flag.
- Usercode of associated job (SPO$US)
- End of file flag (SPO$XX)
- Digits 43-47 of record — value depends on record type code
- Text of ODT message (SPO$LN)

SNAP

This option prints a formatted listing of the SNAP picture contained in the memory dump file. SNAP pictures are inserted into memory under certain program or system failure conditions. The contents of a SNAP picture differs depending on the type of V Series system (V 300, V 500, and so forth).

SUBPORTIO

This option prints the information in the Shared Subport Memory Area (SSMA). There is one SSMA for each pair of open Subports on the system. An area contains the input and output data queues and attributes shared by both connected Subports.

TRAK

This option prints a formatted listing of the TRAK buffer. Each entry in the DMPANL listing represents one TRAK call. DMPANL prints different information depending on the options used in the specific TRAK call. A small set of default information is printed for every TRAK call. This information includes:

- Location of the TRAK entry within the TRAK buffer
- Mix number of task that executed the TRAK call
- TRAK options of TRAK call
- Current environment number
- Address returned to after TRAK call
- Time of TRAK call
- Contents of index registers (including mobile index registers)

Based on the TRAK options of the TRAK call, DMPANL can print:

- Environment number and address of the last 10 stack entries (local calls show zeros as the environment number).

- Stack parameters of the last stack entry (all parameters that were passed to the procedure containing the TRAK call).
- Q1 area or the Q2 area.
- Any data in memory (based on parameters in TRAK call).

TS

This option prints information obtained from or related to the CANDE initialization deck. It includes the Time Sharing Program Information Block (PIB) and the Shared Area Table.

- Timesharing PIB Table
  - Contains one entry for each program contained in the CANDE initialization file (deck).
  - Each entry contains the name, starting disk address, and size (in KD) of the program's object code file. The entry also contains:

    The disk address of one disk directory block. The disk directory header information in the block contains an entry for the program's object code code file.

    The disk address of the disk file header for the code file.

    An index showing the relative position of the code file's entry in the disk directory header information of the relevant disk directory block.
- Shared Area Table

  Contains one entry for each timesharing handler program contained in the CANDE initialization file.

  Each entry contains the name or the handler program and the priorities assigned to the program in the CANDE initialization file. The entry also includes the size of the shared area and the login inhibit flag, both of which are ignored under the MCP/VS operating system.

WAIT

This option prints information about each individual task that is waiting on some condition when the system memory dump is performed. DMPANL obtains some information from the reinstate list and other information from the mix table.

The printed information includes:

- The task number, task state and the current lock number of the waiting task (RI-TSK, RI-SID and RI-MCL in the reinstate list). The current lock number does not represent the *only* lock owned by the task; it represents the lock with the highest cannonical lock number.
- The first seven digits of the task's mix table entry. These digits are known as the wait flags (MIXFLG in the DMPANL listing). If any wait flag contains a value other than zero, DMPANL prints the name of the individual wait flag and a description of the wait condition represented by the value stored in the wait flag. Descriptions of the wait flags and their possible values are included in the mix table definition of the Module Interface Description (MID) of the operating system.

TASK PARAMETERS

Enter task parameters as part of a PM1 command to control what information about an individual task is included in a system memory dump. Figure 3-7 shows the syntax for these parameters.

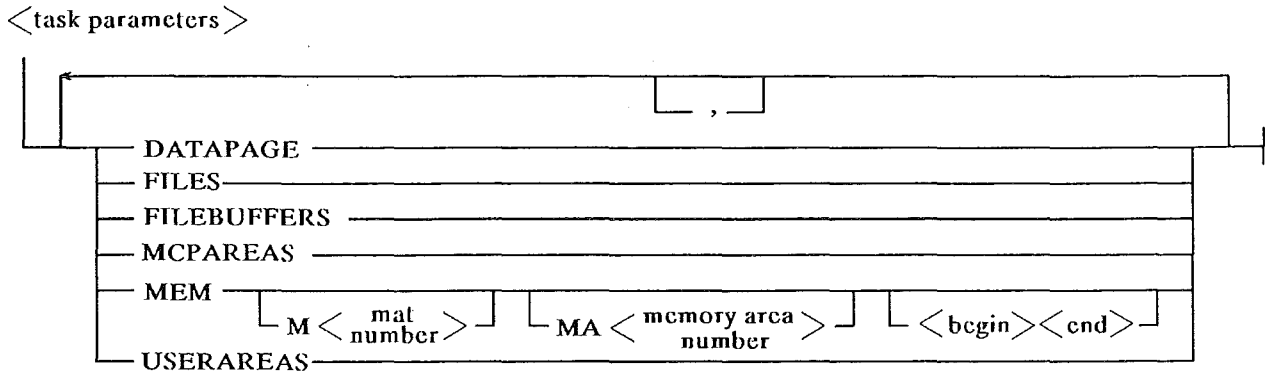


Figure 3-7. Task Parameters (DMPANL)

All task selection parameters must be preceded by the keyword TASK and a <task number> that identifies an individual task that was on the system when the system memory dump was performed. If you enter TASK <task number> without any further parameters, the default task syntax is used. The defaut syntax prints all of the task's memory areas and operating system information that relates to the task.

DATAPAGE

This option prints the individual task's MCP data page.

FILES

This option prints information about all of the individual task's files that are open or that were opened and closed without release. The printed information includes:

- File Information Zone (FIZ)
- External FIB (maintained by operating system)
- Device Assignment Table — also known as Input Output Assignment Table (IOAT)
- File Header (as stored in memory)
- Address blocks
- I/O Queue Elements
- File Information Block (FIB)

FILEBUFFERS

This option prints the same information as FILES, but also prints the external file buffers for any of the individual task's files that have external file buffers.

MCPAREAS

This option prints the individual task's mix table entry and reinstate list entry. Descriptions of the mix table and reinstate list entries are provided under Commonly Used Data Structures later in this section.

MEM

This option prints a raw memory dump of a specific memory area of an individual task. If no parameters are entered after MEM, default syntax is used. Default syntax specifies the first Memory Area Table (MAT) of the individual task, memory area zero (the task's user data area) and beginning and ending memory addresses corresponding to the entire memory area.

- **M <mat number>**: The number of one of the individual task's Memory Area Tables (MAT). For object code files generated with pre-V Series compilers, the only allowed MAT number is 1 (the default).
- **MA <memory area number>**: The number of one of the memory areas within the specified Memory Area Table (MAT). Memory area number can range from 0 to 7. (The default is 0.)
- **<begin><end>**: The beginning and ending memory addresses of a portion of memory within the specified memory area of the individual task. Memory addresses are entered in KD and the default includes the entire memory area.

USERAREAS

This option prints only the user's data and code.

COMMONLY USED DATA STRUCTURES (DMPANL FORMATS)

The MCP/VS operating system uses many different data structures (sometimes called "tables"). Any one of these structures can be important in a given support situation, but several of them are important in *many* different situations. The tables on the following pages describe the DMPANL listings of several frequently used operating system data structures.

These tables list each field in the DMPANL listing give a brief description of the field's contents. The sizes and types of the fields are not listed. The Module Interface Description (MID) of the operating system provides more details about the structures described here. The *V Series MCP/VS Architecture Manual* describes the purpose and use of each data structure.

The data structures described on the following pages include:

- Reinstate List
- Mix Table
- I/O Queue Elements

Reinstate List

Table 3-1 describes the individual fields included in the DMPANL listing of a reinstate list entry.

Table 3-1. DMPANL Listing of Reinstate List Entry

DMPANL Field Name	Description
link_ by-_ priority	Pointer to next reinstate list entry. Runnable entries are linked in order based on priority (operating claim). Dozing entries are linked in order by next scheduled run time.
RI-FLT	Soft fault pending flags. Includes forced timer interrupt and asynchronous message waiting flags.
RI-OTK	Failed lock area, lock owner number. (Mix number of the task that owns the lock that this task failed to obtain.)
RI-NSR	Next scheduled run time for this task
RI-#ET	Environment table size
RI-FGS	I/O Flags. Reinstate list bit flag. Includes timer interrupt fault, driver task, error I/O independent runner, and normal I/O independent runner flags.
RI-WTK	Failed lock area, next task in list. (Mix number of the next task on the list of tasks that are waiting for the lock that this task failed to obtain.)
RI-WAT	Increment to next scheduled run time for this task.
RI-FLG	Task type. Possible values include user task, driver task, timesharing task-0, MCS task, timesharing task-1, DBP task, or independent runner.
RI-PR	Task processor priority
RI-MCL	MCP cannonical lock number
RI-DIR	Direct time accumulated by this task
et_ expan-sion_ area	Environment table expansion area. Present for future considerations.
RI-SID	Task state indicator. Possible values include runnable, waiting lock, waiting event, dozing, waiting termination, failed hardware call, kernel entry, waiting start up, available, suspended, or waiting driver.
RI-UCL	User cannonical lock number
RI-TSR	Time slice remaining
RI-ET	Environment table MCP-relative memory address
RI-SOV	Stack overflow flag
RI-TSK	Task number (mix number) of this task
RI-R/D	R/D from failed hardware call.
RI-MEM	Task memory priority
RI-OPC	Task operating claim

Table 3-1. DMPANL Listing of Reinstate List Entry (Continued)

DMPANL Field Name	Description
next_ increment_ in_ time_ slice	Amount assigned as task time slice if task is interrupted by a timer interrupt. Used in calculating RI-NSR.
RI-SMD	Saved mode toggles
misc task flags	Reinstate list bit flag. Includes doze until time flag.
	INTERRUPT FRAME
RI-ACM	Accumulator
RI-MRG	Measurement register
RI-MOD	Mode toggles
RI-AE#	Active environment number
RI-MIX	Mobile index registers
RI-MSK	Interrupt mask
RI-COM	Comparison indicators and overflow indicators
RI-ADR	Next instruction address

Mix Table

Table 3-2 describes the individual fields included in the DMPANL listing of a mix table entry.

Table 3-2. DMPANL Listing of Mix Table

DMPANL Field Name	Description
	WAITING INFORMATION
MIX-LK	Link to next mix table entry. Entries are linked in order of task initiation.
MIX-IO	Mix table waiting flag — Waiting I/O processing. Some possible values include: Not waiting I/O Read/write I/O complete File close queue flush Stoppage No space in complex wait table Positioning DCOM fill enabled Program overlay Waiting complex wait Terminate queue flush Waiting LIO open Waiting LIO close Waiting LIO interrogate Waiting program call Waiting no ready device
MIX-WM	Mix table waiting flag — Waiting module processing. Some possible values include: Not waiting module Waiting DCOM BCT processing Waiting core to core (CRCR) Waiting BNA module Waiting STOQUE data entry Waiting STOQUE memory Waiting STOQUE name slot Waiting trace Available wait Waiting TSM handler transfer Waiting DCPC

Table 3-2. DMPANL Listing of Mix Table (Continued)

DMPANL Field Name	Description
MIX-OK	Mix table waiting flag — Waiting operator or operating system action. Some possible values include: Not waiting operator Duplicate file Duplicate library on disk No user disk available No disk file Locked disk file No "non-disk" file Output device required Extension module not in memory
MIX-CR	Mix table waiting flag — Waiting memory. Currently only :1 bit is used. Some possible values include: Not waiting memory Waiting disk backup open Waiting Direct I/O open Waiting DCP open Waiting DCOM open Waiting Port/Subport open or set
MIX-WA	Mix table waiting bit flag. Includes push in process, program stopped, waiting MICR I/O complete, and DCP activity flags.
MIX-WB	Mix table waiting bit flag. Includes task sleeping, trace print in process, trace printer required, and trace disk required flags.
MIX-WC	Mix table waiting bit flag. Includes TSM job waiting terminal I/O complete, program suspended, waiting mix, and waiting direct I/O
MIX-SP	Schedule priority.
MIX-MP	Memory priority.
MIX-RP	Processor priority.
MIX-IP	I/O priority.
MIX-CP	Core to core name
MIX-CO	Core to core status flag
MIX-DA	Core to core data address
MIX-DL	Core to core data size
MIX-CK	Pointer to core to core list
MIX-DZ	Pointer to doze list

Table 3-2. DMPANL Listing of Mix Table (Continued)

DMPANL Field Name	Description
	PROGRAM IDENTIFICATION
MIX-ID	Program ID
MIX-MF	Multiprogram ID
MIX-RQ	Task number of job requesting PCR
MIX-RJ	RJE originator key
MIX-RL	Run Log ID number
MIX-PI	Program initiation code
MIX-PG	Mix table bit flag. Designates special types of programs, including DMS control, WFL-type handler, BNA handler, DMPALL and so forth.
	SCHEDULE INFORMATION
MIX-SS	Schedule status digit
MIX-PL	Precedence link
MIXAID	After linkage program ID
MIXAMF	After linkage multiprogram ID
MIX-VA	VALUE address
MIX-VL	VALUE length
MIX-VD	VALUE data
MIX-GT	Trace parameters
	REINSTATE INFORMATION
MIX-RN	Program execution flags
MIX-TG	Mode overflow condition toggle
MIX-NO	Mix number of task
MIXSFD	Pointer to disk directory header block (GO phase of COMPILE & GO operation)
MIXSX1	Index into disk directory header block (GO phase of COMPILE & GO operation)
MIX-ME	Program size in pages
	SUPPLEMENTARY WAIT INFORMATION
MIX-XX	Supplementary wait information. Contents dependent on waiting flags. Data can represent IOAT address of working file, memory required, dcp number, time until wake from doze, time waiting IOC initiated, or serial number of required diskpack.
MIX-IH	Datacomm wait pointer

Table 3-2. DMPANL Listing of Mix Table (Continued)

DMPANL Field Name	Description
MIX-HW	Hardware type waited for
MIX-HI	HIHO status digit
MIX-HO	HIHO status digit
MIXWTC	Unused
	PROGRAM STATUS INFORMATION
MIX-IC	IOATs assigned to task
MIX-FC	Number of files declared
MIX-SG	Number of disk segments in program
MIX-FW	Process call forward link
MIX-BW	Process call backward link
MIX-PC	Mix number of process caller
MIX-TC	Terminate code
MIX-DS	User DS code
MIX-FP	File parameter flag
MIX-DD	Mix table bit flag. Includes task initiated through ZIP, executed with lock, initiated from PCR and change number supplied flags.
MIX-LQ	Mix table bit flag. Includes memory dump abnormal termination, Direct I/O use incremented, time limit supplied and label equate supplied flags.
MIX-SR	Mix table bit flag. Includes program sorting, FLAME, programm call and SORT. flags.
MIX-DC	Mix table bit flag. Includes simulated processor interrupt, midnight overlap for doze, disallow operator breakout and use procedure in process flags.
MIX-TR	Mix table bit flag. Includes terminate running, call terminate at end of dump breakout, pass breakout records if ARMed and breakout not allowed flags.
MIX-FL	Mix table bit flag. Includes accept/display messages on disk, codefile on diskpack, RJE origination and compilation to diskpack flags.
MIX-PR	Mix table bit flag. Includes parameter passed, handler program identification and independent runner identification flags.
MIX-FG	Mix table bit flag. Includes bound intrinsic, TEST, DEBUG and RERUN identification flags.
MIX-TP	Mix table bit flag. Includes programs tracing, trace heading required, TSM main mix and close in process flags.
MIX-TM	Mix table bit flag. Includes terminate schedule in process, MCP workfile in use by task, no codefile at terminate and syntax error flags.

Table 3-2. DMPANL Listing of Mix Table (Continued)

DMPANL Field Name	Description
MIX-FF	Mix table bit flag. Includes user program required, MCP intrinsic required, GO phase label equate disk obtained (COMPILE & GO operation) and compile/execute phase label equate disk obtained (COMPILE & GO operation) flags.
MIX-EU	Disk ID of code file
MIX-CF	Familyname of code file
MIXSTQ	STOQ module in use
MIXMCR	MICR module in use
MIXTRC	TRACE in use
MIXDCM	DCOM module in use
MIXAUS	Mix table bit flag. Descibes the asynchronous commands that have been entered for this job.
	TIMING INFORMATION
MIX-TL	Time limit remaining in seconds
MIX-CC	Number of child tasks of this task
MIX-PM	Parent task of this task
MIX-PT	Accumulated pro-rated time
MIX-IT	Accumulated "Waiting IOC" time
MIX-AV	Average run wait time
MIX-BJ	BOJ time in seconds
MIX-TI	Total stopped time in seconds
MIX-OT	Task overtime
	SECURITY INFORMATION
MIX-ET	Mix table bit flag. Includes user table entry made, user file maintenance possible, user card entry made and reserved for system access security flags.
MIX-UC	User code of initiator
MIX-LV	Access level (SPO level) of initiator
MIX-PV	Library maintenance allowed, Direct I/O allowed.
MIX-CL	Security type
MIX-CN	Charge number
MIX-SU	Security use
MIX-SN	Sensitive data flag

Table 3-2. DMPANL Listing of Mix Table (Continued)

DMPANL Field Name	Description
MIX-GD	Security guard
MIX-FA	Security family
	SPECIAL FILE INFORMATION
MIX-PB	Disk address of program parameter block
MIX-FB	Disk address of file parameter block
MIX-FH	Address of program's disk file header
MIX-FD	Address of disk directory header block containing disk directory header entry for program
MIX-FX	Index into disk directory header entry within disk directory header block
file_ equate _ offset	Offset into file equate block in memory
	TIMESHARING INFORMATION
MIX-SA	Name of shared area
MIX-BF	Buffer address in handler
MIX-CD	Mix table bit flag. Includes low core initialization inhibited, type 2 process and code file in memory flags.
	MISCELLANEOUS INFORMATION
MIX-FE	Label equate list pointer
MIX-EV	Mix table event
MIXTRM	Termination OK
MIXTCT	Terminate stoppers
MIX-FT	Originator fault
MIXTLK	Task must talk
	WY PROCESSING INFORMATION
MIX-CH	Current hardware type
MIX-RH	Remote host name
MIXSGR	Segments required
MIXPCM	Psuedo card reader message serial number.

Table 3-2. DMPANL Listing of Mix Table (Continued)

DMPANL Field Name	Description
	DEBUG INFORMATION
MIXCTK	DEBUG controlling task number
MIXDBG	DEBUG flags
MIXCWT	Complex wait flags
MIXWFL	WFL-type process flag
MIX-UT	Waiting not ready device pointer
MIXWXC	Waiting expand core moemory
MIX-NL	NL command permitted flag
MIXOKD	OK for dump file
MIXHWD	Hardware type for dump file

I/O Queue Elements

Table 3-3 describes the fields included in the DMPANL listing of an I/O queue element.

Table 3-3. DMPANL Listing of I/O Queue Elements

DMPANL Field Name	Description
Q-LINK	Link to next I/O queue element in the "waiting" I/O queue. (Offset within queue element memory area).
Q-TASK	Task number of task requesting I/O
Q-LIOF	Logical I/O Flags. Includes user queue element present, LIOM complete not done, and R/D posted to buffer status word at I/O complete flags.
Q-TPRI	Priority of task initiating I/O
Q-SYST	Pointer to system queue element. Cross-link between system queue element and user queue element.
Q-HDW	Hardware type
Q-INH	Ignore inhibits digit
Q-SPRI	Saved queue priority. Not currently used
Q-WRIT	Address of associated WRITE queue element
Q-EUAD	Pointer to EU table
Q-TYPE	Special handling digit. Includes recovery operation, status test wait ready, and fire on indicated channel flags.
Q-IOCR	Address offset used to index into a routine within the I/O module (IOM). The routine is named IOCR and updates the I/O block count (IO-BCT).
Q-EXCH	Logical exchange number
Q-IOAT	Address of IOAT within File Information Area (FIA) or the system tables.
Q-RCVY	Recovery required digit. Not currently used. Included maintenance log record required, reset inhibits on good I/O, subsequent initialization of I/O, and no MLOG required flags.
Q-CNVT	Address offset used to index into a routine within the I/O module (IOM). The routine is named CNVT and sets up the C address of an I/O based on the requirements of the particular DLP (decimal or binary addressing).
Q-CHAN	Channel I/O was initiated on
Q-BSW	Address of buffer status word
Q-CLR	Clearing digit. Not currently used. (On successful I/O complete, this digit was ORed over Q-TYPE to clear unwanted bits).
Q-EVNT	I/O complete event structure
Q-ODLP	Original channel used
Q-TSTA	Address of task status digit

Table 3-3. DMPANL Listing of I/O Queue Elements (Continued)

DMPANL Field Name	Description
Q-LOCK	Shared lock flag. Includes force error-free I/O complete, shared I/O complete service required, and bit entry attached flags.
Q-IOTM	I/O time for this I/O
Q-UNIT	Unit number of device
Q-USTA	Address of unit status digit
Q-SFG4	Saved error handling flags. Not currently used.
Q-IOCT	I/O complete event time
Q-ERCT	Error retry count
Q-BDMB	Beginning address of buffer dump
Q-FLG4	Error handling flags. Includes no recovery action, mask add inhibited, keyboard error display inhibited, and ignore unrecovered error flags.
Q-SPRT	Special I/O complete routine link
Q-IOQ#	I/O queue element count. Not currently used.
Q-BDME	Ending address of buffer dump
Q-MASK	Error handling mask
Q-RSZ	Maximum record length
Q-MBS	Maximum block size
Q-BLK	Blocking technique. Not currently used.
Q-FLAG	Queue element flags
Q-RSRV	Reserved field (I/O Module)
Q-VI/O	Virtual I/O descriptor
Q-R/D	Virtual result descriptor (R/D)
Q-EVRD	DLP recovery result descriptor 10. Not currently used.
Q-RI/O	Relative I/O descriptor
Q-NR/D	DLP result descriptor
Q-ENRD	DLP recovery result descriptor 16
Q-RCIO	Result of CIO instruction
Q-XNRD	DLP result descriptor for sequential DLP
Q-XENR	DLP recovery result descriptor for sequential DLP
Q-IOC	Result of IOC instruction

SECTION 4
SUMMARIES OF STATE —CHANGING INSTRUCTIONS

The *V Series MCP/VS Architecture Manual* explains the architecture of the MCP/VS operating system in detail. It is important to understand the material in the *V Series MCP/VS Architecture Manual* before working with the operating system.

As a supplement to the *V Series MCP/VS Architecture Manual*, this section provides brief summaries of several important V Series instructions. These summaries provide a conceptual, simplified description; they do not explain or instruct. They are designed for quick reference *after* the operating system is understood. Detailed information is not included in the summaries. For detailed information, see the *V 300 System Reference Manual* or the V Series Instruction Set System Design Specification.

The instructions in this section have the potential to change the state (memory environment) of the system when they are executed. They involve some of the most important and distinctive aspects of the MCP/VS operating system.

The instructions summarized here include:

- Hypercall (OP 62)
- Branch Communicate (OP 30)
- Virtual Enter (OP 35)
- Return (OP 63)
- Branch Reinstate Virtual (OP 93)

HYPERCALL INSTRUCTION (HCL - OP 62)

The Hypercall instruction (HCL) is the replacement for the Branch Communicate Instruction (BCT, OP 30) used under MCPIX. The HCL transfers control from one environment to another. A common example is from a user program to the operating system. Detailed information about the HCL instruction is provided in the *V 300 System Reference Manual.*

The main portions of memory involved in the execution of an HCL instruction are:

- The memory area containing an individual task's code (environment 1, memory area 1). Under the current version of MCP/VS, this is a copy of environment 0, memory area 1, but this relationship may change in subsequent releases of the operating system.
- The memory area containing an individual task's data (environment 1, memory area 0). Under the current version of MCP/VS, this is a copy of environment 0, memory area 1, but this relationship may change in subsequent releases of the operating system.
- The MCP Data Page of an individual task.
- The operating system Environment Table (ET).
- The Memory Area Table (MAT) of the selected portion of the operating system.
- The memory area containing the code of the selected portion of the operating system.

Some other system tables are indirectly involved, but are omitted to keep this summary simple.

Hypercall Format

62 AFBF <A address> <B address>

AFBF

Defines the length, in bytes, of the HCL parameters pointed to by the <A address>.

<A address>

Points to the HCL parameters (if any). These parameters are stored on the individual task's stack which in turn is stored in the task's MCP Data Page.

<B address>

Points to the HCL function number in the memory area containing the individual task's data.

Hypercall Operation

- The HCL function table is located in the individual task's MCP Data Page, allowing the HCL instruction to be reentrant. The processor firmware maintains a pointer to allow access to the active task's User Services Memory Area Table (USMAT). The HCL instruction uses the HCL function number as an index to the correct entry in the USMAT.
- The environment number of the new environment is taken from the HCL function table and is used as an index into the operating system Environment Table (ET).
- The entry in the operating stystem ET points to the Memory Area Table (MAT) of the appropriate portion of operating system code. This MAT provides full addressability to the new environment.
- The HCL stack frame is built on the individual task's MCP Data Page. This stack frame includes the old active environment number and return address. The HCL parameters are transferred from the memory area pointed to by the <A address> to the stack frame on the task's MCP Data Page.
- The machine state is reset according to data found in the HCL function table (including the next instruction address) and processing is resumed at the next instruction in the new environment.

To return control to the original environment, the RET instruction is used. RET is the complement of the HCL instruction, the VEN instruction and the Hardware Call procedure.

BRANCH COMMUNICATE INSTRUCTION (BCT - OP 30)

The Branch Communicate (BCT) instruction lets a user program request the operating system to perform a service. The effect of this instruction is the same under MCP/VS as under MCPIX, but the way the instruction works internally under MCP/VS is completely different from MCPIX.

Under MCP/VS, the BCT instruction operates very much the same as the Hypercall instruction (HCL, OP 62). The main difference between the two instructions is the way that parameters are passed. The BCT parameters are located in the memory area containing an individual task's code, while the HCL parameters are located in the individual task's MCP Data Page.

The BCT instruction uses the same portions of memory as the HCL instruction with the exception of the memory area containing the individual task's data. The portions of memory used by the BCT instruction are:

- The memory area containing an individual task's code (environment 1, memory area 1). Under the current version of MCP/VS, this is a copy of environment 0, memory area 1, but this relationship may change in subsequent releases of the operating system.
- The MCP Data Page of an individual task.
- The operating system Environment Table (ET).
- The Memory Area Table (MAT) of the selected portion of the operating system.
- The memory area containing the code of the selected portion of the operating system.

Although the BCT instruction does not access the memory area containing a task's data, the BCT instruction has *implicit* access to this memory under the current release of MCP/VS. This is because (under the current release) the memory area containing a task's data and the memory area containing a task's code are physically in the same portion of memory. This relationship may change in subsequent releases of the operating system.

Branch Communicate Format

```
30 AFBF
   <branch instruction>
   <BCT parameters>
```

AFBF

Contains an index into the BCT function table in the MCP Data Page of the individual task.

<branch instruction>

An instruction that branches around the <BCT parameters> that follow.

<BCT parameters>

Information stored in the memory area containing the individual task's code that control the BCT instruction.

Branch Communicate Operation

- The environment number of the new environment is taken from the BCT function table and is used as an index into the operating system Environment Table (ET).
- The entry in the operating system ET points to the Memory Area Table (MAT) of the appropriate portion of operating system code. This MAT provides full addressability to the new environment.
- An HCL-type stack frame is built in the individual task's MCP Data Page, including the old active environment number and return address. The BCT parameters are transferred from the memory area containing the individual tasks's code onto the stack.
- The machine state is reset according to data found in the BCT function table entry (including next instruction address). Processing is resumed at the next instruction in the new environment.

To return from operating system memory environment to the individual task's memory environment, the RET instruction is used.

VIRTUAL ENTER INSTRUCTION (VEN - OP 35)

The Virtual Enter instruction (VEN) transfers control to another portion of code within the current memory environment or a different memory environment of the current task. Parameters are passed to the succeeding code on the stack. A VEN within the current memory environment is called a local VEN. A VEN to a different memory environment of the current task is called a non-local VEN.

The VEN instruction works much the same as the Enter instruction (NTR), but the VEN parameters are stored in the MCP Data Page of the individual task, while the NTR parameters are stored in the memory area containing the individual task's code. (The NTR instruction can only transfer control within its own environment.)

The VEN instruction is also similar to the Hypercall instruction (HCL). The main difference is that HCL implicitly selects a new data page, the VEN remains working with the same data page.

The memory areas involved in the execution of the VEN are:

- The Environment Table (ET) of the individual task.
- The Memory Area Table (MAT) of the current memory environment (local VEN)
- The MAT of the memory environment that control will be transferred to (non-local VEN)
- The User Services Memory Area Table (USMAT)
- The MCP Data Page of the individual task.
- The memory area containing the currently executing code (local VEN).
- The memory area containing the code that control is transferred to (non-local VEN).

Virtual Enter Format

35 AFBF <A address> <B address>

AFBF

Defines the length, in bytes, of the VEN parameters.

<A address>

Points to the VEN parameters (if any) in the MCP Data Page of the individual task.

<B address>

Points to a 20 digit environment field, which specifies the new environment (if any) and the next instruction address within that environment. This environment field is stored in the memory area containing the individual task's code.

Virtual Enter Operation

- The VEN Environment field is located in the memory area containing the individual task's code.
- If the environment number is zero, the VEN is local, and no new environment needs to be loaded. If the environment number is non zero, it is used as an index into the current environment table. The new environment is loaded, with addressability to the current data page maintained because of the convention that the new MAT entry zero resolves the same way as the previous MAT entry zero.
- The VEN stack frame is built in the Data page, including the return Environment number (if applicable) and return address. The VEN parameters are transferred to the stack.
- Processing is resumed at the new location specified in the Environment field.

Return to the instruction following the VEN is accomplished by use of the RET instruction.

RETURN INSTRUCTION (RET - OP 63)

The Return instruction (RET) returns control from one portion of the operating system back to:

- Another portion of the operating system, running for an individual task, or
- An individual task's code.

RET complements several instructions or procedures. Control may be transferred to the current portion of the operating system by a Hypercall (OP 62), Branch Communicate (OP 30) or Virtual Enter (OP 35) instruction. RET reverses the effect of all these instructions. RET also returns control after control is transferred through a Hardware Call procedure.

In all cases, the return instruction address, and the environment number of that address are located on the stack in a specific type of stack frame. The stack frame type changes depending on which instruction or procedure transferred control. RET acts differently depending on the type of stack frame, but the result is the same in each case.

Control can be returned to the same or a different memory environment.

If control is returned to an individual task's code, the memory areas involved in the execution of RET are:

- The MCP Data Page of the individual task (which contains the stack).
- The Reinstate List Entry of the individual task.
- The Environment Table (ET) of the individual task.
- The Memory Area Table (MAT) containing the individual task's active memory area.
- The memory area containing an individual task's code (environment 1, memory area 1). Under the current version of MCP/VS, this is a copy of environment 0, memory area 1, but this relationship may change in subsequent releases of the operating system.

Return Format

63 AFBF

AFBF

Unused and reserved.

Return Operation

This example describes an RET instruction returning control from the operating system (running for an individual task) back to the individual task's code. This RET returns control that has been transferred by a BCT (OP 30) instruction.

- The RET instruction obtains an index into the task's environment table (ET) and a next instruction address from the stack frame. Other information is also obtained (mode toggles, interrupt mask, accumulator and so forth).
- RET examines the ET index obtained from the stack frame.
  - If the most significant digit of the ET index is a **D**, control is being returned to another part of the operating system that is running for the individual task. This does not apply to the current example.
  - If the most significant digit of the ET index is a value from **0** to **9**, control is being returned to an individual task. RET uses the task number as an index into the reinstate list.
- RET reads the reinstate list entry of the individual task and obtains the location of the task's environment table (ET).
- RET uses the ET index obtained from the stack and the ET location obtained from the reinstate list to read one entry in the task's ET. This entry contains the location of one of the task's Memory Area Tables (MATs)
- RET uses the MAT location obtained from the ET find the task's active MAT and reads the first eight entries in it.
- RET loads memory area 1 of the active MAT and begins execution at the next instruction address within that memory area.

BRANCH REINSTATE VIRTUAL INSTRUCTION (BRV - OP 93)

The Branch Reinstate Virtual instruction (BRV) establishes the memory environment of a given task and transfers control to that task. The BRV instruction is used by the kernel of the MCP/VS operating system to dispatch other tasks. It is the only way that control can pass out of the kernel. (The BRV instruction under MCP/VS is similar to the BRE instruction under MCPIX.)

BRV can be considered as the complement of the interrupt procedure. The interrupt procedure transfers control into the kernel, while BRV transfers control out of the kernel to individual tasks.

The memory areas involved in the execution of BRV are:

- The Reinstate List Entry of the task being dispatched. This entry contains the interrupt frame of the task and the next instruction to execute within the task.
- The Environment Table (ET) of the task being dispatched. This can be the operating system's ET or the ET of a task.
- The MCP Data Page of the kernel.
- The Memory Area Table (MAT) of the task being dispatched. This can be the MAT of a part of the operating system that is running for a task or the MAT of the task itself.
- The memory area containing the code of the task being dispatched.

Branch Reinstate Virtual Format

93 AFBF

AFBF

Unused and reserved.

Branch Reinstate Virtual Operation

- The BRV instruction looks at IX1 in the kernel's MCP Data Page. IX1 contains an index into the reinstate list. This index determines which task will be reinstated.
- BRV reads the reinstate list entry pointed to by IX1. From this entry it obtains the interrupt frame of the task and the next instruction within the task to execute. One portion of the interrupt frame contains an index specifying one entry in the Environment Table (ET) of the task being dispatched.
  - If the most significant digit of the ET index is a value from **0** to **9**, the task being dispatched is a user task. BRV reads an additional portion of the reinstate list entry pointed to by IX1 and obtains the location of the task's ET.
  - If the most significant digit of the ET index is a **D**, the task being dispatched is a part of the operating system that is running for a task. BRV now reads entry 0 in the reinstate list (a special entry for the operating system) and obtains the location of the operating system's ET.
- BRV uses the location of the appropriate ET and the ET index obtained from the interrupt frame to read one entry in the ET. This entry contains the location of a Memory Area Table (MAT). (Either the task's MAT or the MAT of one portion of the operating system.)

- BRV reads the MAT and uses the first 8 entries of the MAT to create the memory environment of the task being dispatched.
- BRV instructs the hardware to begin processing at the next instruction address (obtained from the interrupt frame) within memory area 1 of the dispatched task's memory environment. Memory area 1 always contains the task's executable code.

SECTION 5
INPUT/OUTPUT SUMMARY

This section summarizes the input/output (I/O) functions of the MCP/VS operating system. Parts of this section assume that you are familiar with I/O operation under the MCPIX operating system (predecessor to MCP/VS). These places are clearly designated. The best sources for information about I/O under the MCPIX operating system are customer education or Unisys field support training documents.

This section is organized as an example. It follows the sequence of events within the operating system as a user program opens a file and reads information from it. The example concentrates on logical I/O operation; it does not describe other parts of the operating system in any detail. Within the description, when the MCP/VS operating system performs an action the same way the MCPIX operating system does, detailed material is omitted.

As you follow the example, use the memory allocation diagram in figure 5-1 for reference. This diagram shows the major data structures involved in I/O operations and the links between them. Some links shown in the diagram do not apply to the first part of the example because the links are created as the file is opened, but the diagram is a good reference for the example as a whole.

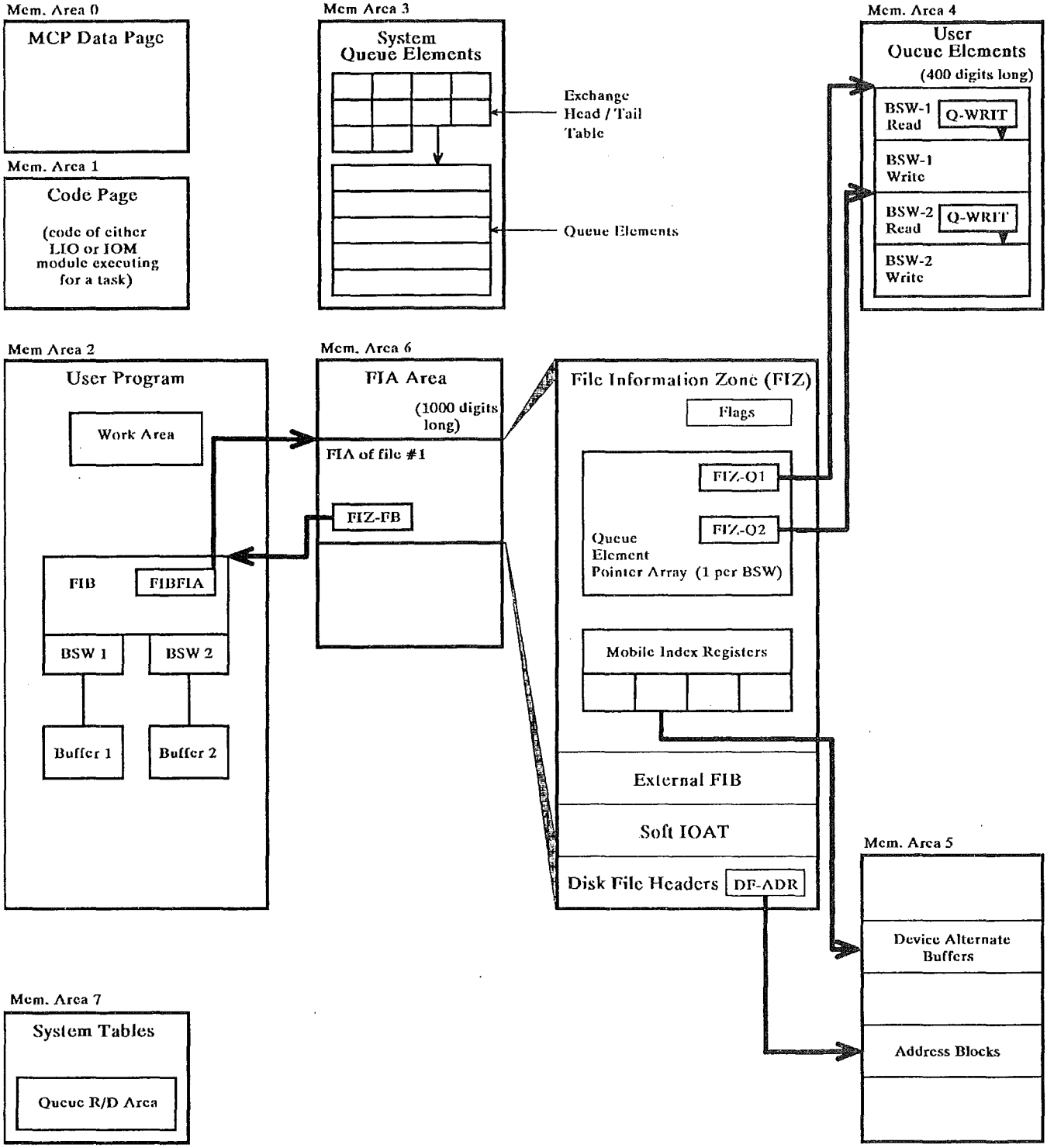


Figure 5-1. Operating System Memory Allocation After OPEN

I/O EXAMPLE

The following paragraphs describe the sequence of events within the operating system as a user program opens and reads information from a disk file.

1. A user program issues an OPEN BCT to instruct the operating system to open a given file.

   a. The Open module (OPN) of the operating system has access to:

      - The MCP Data Page of the task requesting the open (memory area 0)
      - The MCP Code Page (memory area 1)
      - The code and data of the user program requesting the open (memory area 2)
      - The System Tables (memory area 7)

   b. The OPN module calls the Memory Manager module (MMG) to request memory for the File Information Area (FIA). After obtaining the memory, OPN creates the FIA entry for the file being opened. The FIA entries are created in memory area 6 of the task's memory environment.

      The basic parts of a FIA entry are:

      - File Information Zone (FIZ)
      - External FIB (External File Information Block)
      - IOAT entry
      - Disk File Header (as stored in memory)

      OPN sets up a pointer in the internal FIB of the user program that points to the FIA entry of the file. Under the MCP/VS operating system, this pointer is named FIBFIA (under MCPIX, it is named FIBIOA). Within the FIA entry, OPN sets up a pointer (named FIZ-FB) that points back to the internal FIB of the user program.

   c. The OPN module calls MMG to request memory for the MCP Heap Area (memory area 5 of the task's memory environment). After obtaining the memory, OPN creates the address blocks for the file and sets up a pointer in the disk file header of the FIA to point to the address blocks.

      If the file is a backup file, OPN creates the device alternate buffers and sets up a pointer in the mobile index registers of the FIZ to point to the device alternate buffers.

   d. The OPN module calls MMG to request memory for the user queue elements (memory area 4 of the task's memory environment). After obtaining the memory, OPN calls a routine in the Logical I/O module (LIO) to create the user queue elements.

      The routine in the LIO module is named BLDQUE. BLDQUE creates the user queue elements. BLDQUE also sets up the pointers in the queue element pointer array of the File Information Zone (FIZ) so that they point to their corresponding user queue elements.

      BLDQUE creates one or two user queue elements for each buffer declared in the internal FIB of the user program (in the FIBALT field). Because of this, the number of buffers cannot be increased after the file is opened.

e. The OPN module sets up pointers in the FIZ (named FIZ-RD, FIZ-WT, FIZ-SK and FIX-PN) so that they point to tailored I/O routines within the LIO module. These tailored I/O routines vary depending on the type of device that the file is stored on and the type of open requested by the user program (Input, Input/Output and so forth).

f. If the file is accessed as a sequential input file, the OPN module issues buffer fills.

g. The OPN module finishes execution and control returns to the user program.

2. The user program issues a READ BCT to instruct the operating system to obtain information from the file. The READ BCT begins executing at an entry point in the Logical I/O module (LIO).

   a. The LIO module checks for a valid READ request (checks for open file, valid internal FIB within user program, valid pointers between the internal FIB and the FIA entry, and so forth).

   b. The LIO module branches to the appropriate tailored I/O routine based on the address contained in the FIZ-RD field of the file's FIZ.

   c. The tailored I/O routine within LIO determines if a physical I/O is needed to obtain the information.

      The MCP/VS operating system makes this decision in the same way as the MCPIX operating system. There are no major differences in the way the two operating systems handle deblocking and buffer rotation.

   d. If a physical I/O is needed to obtain the information, the LIO module calls a routine in the physical I/O module (IOM). The MCP/VS operating system handles this process differently than the MCPIX operating system.

      MCPIX

      The LIO module assembles an I/O queue element in the Q1 area, using information from the buffer status word, address blocks, disk file header, and so forth. LIO then branches to a routine named IOQ.

      The IOQ routine finishes creating the queue element, moves it from the Q1 area into the linked list of queue elements, and falls into a routine named IWOR that fires the I/O.

      MCPIX only maintains queue elements for I/O requests that are in process or waiting to be fired (queued). No queue elements are maintained for idle buffers.

      MCP/VS

      The LIO module assembles a *small* amount of the queue element at the time the I/O is requested. Most of the user queue element was built at the time the file was opened. When the LIO module determines that a physical I/O is needed, roughly 3/4 of the queue element is already created. LIO updates only the 12-digit disk sector address, the unit number (on some occasions), and the memory address of the user program's buffer.

      LIO then calls one of two routines in the IOM module that eventually fire the I/O. These two routines are named BLDFIR (build and fire) and I/OFIR (I/O fire).

      For any file that is opened to allow both input and output, MCP/VS maintains two user queue elements for each buffer status word, (one for reading, one for writing). This eliminates the need to re-create queue elements for alternate reads and writes.

3. The physical I/O module (IOM) performs the physical I/O to obtain data from the disk file.

   a. Depending on whether the unit number in the user queue element must be changed, the logical I/O module (LIO) calls BLDFIR or I/OFIR. (BLDFIR is called when the unit number must be updated.)

      BLDFIR

      This routine recreates the user queue element's exchange number, updates the relative I/O descriptor using the virtual I/O descriptor provided by LIO. The routine then branches to I/OFIR.

      I/OFIR

      This routine converts the 12-digit disk sector address provided by LIO into the appropriate format and then continues with other processing to fire the I/O.

   b. The IOM module (I/OFIR routine) executes a CIO instruction to convert the relative I/O descriptor into a physical I/O descriptor. IOM marks the memory area containing the user program's buffer so that the memory manager module (MMG) does not move the memory area until the I/O completes. IOM then moves the user queue element into the system queue element area (memory area 3 of IOM's memory environment).

   c. The IOM module uses the exchange number in the system queue element as an index into the Exchange Head/Tail Table in the system queue element memory area.

   d. The IOM module locks the lock structure of the exchange.

   e. The IOM module examines the Queue R/D Area in the system tables memory area (memory area 7 of the task's memory environment). IOM looks for a free channel on the desired exchange. If a channel is busy, the Queue R/D entry contains a pointer to the queue element of the I/O in progress on the channel. If the channel is free, the in progress pointer of the Queue R/D entry is zero.

      Free Channel

      IOM inserts a pointer to the current system queue element into the Queue R/D entry of the channel.

      IOM executes an IIO instruction to fire the I/O.

      IOM releases (unlocks) the lock structure of the exchange.

      IOM finishes execution and control returns to the Logical I/O (LIO) module.

      No Free Channel

      IOM links the current system queue element into the linked list of system queue elements that are waiting for the desired exchange. The current queue element is inserted into the linked list based on the task's priority. The linked list is maintained in the system queue element area (memory area 3 of IOM's memory environment).

      IOM releases (unlocks) the lock structure of the exchange.

      IOM finishes execution and control returns to the Logical I/O (LIO) module.

The I/O firing procedure described in this step is very similar to the I/O firing routine (IWOR) of the MCPIX operating system. The major difference are the steps involving lock structures.

4. The physical I/O (IOM) module returns control to the logical I/O (LIO) module. The LIO module is finished dispatching the I/O. LIO always dispatches a type of I/O known as "fire no wait". (There *is* another type of I/O called "fire and wait", but it is normally used by other operating system modules that communicate directly with IOM.)

   LIO returns control to the user program. The user program may wait until an I/O complete occurs for the I/O just dispatched. Whether the user program waits depends on the timing of other processing on the system and on the user program itself. If the user program waits, it is marked as waiting for the event in the user queue element to be signalled.

5. I/O Complete Processing.

   During all system operation, part of the physical I/O (IOM) module executes as an independent runner task on the system. After the DLP successfully places the information from the storage device (in this case, disk) into the buffer within the user program, an I/O complete interrupt occurs. At this point, the kernel of the operating system activates one of two independent runners. These independent runners are:

   - Normal I/O complete independent runner (named IOT). Activated for normal I/O interrupts.
   - I/O error independent runner (named I/OERR). Activated for exception I/O interrupts.

   Throughout the following paragraphs, these two independent runners are referred to collectively as "the I/O independent runner".

   When activated, the I/O independent runner performs the following actions:

   a. The I/O independent runner examines the queue R/D area in the system tables to determine which channel has generated the I/O complete.

   b. The I/O independent runner locks the lock structure of the exchange that has generated the I/O complete.

   c. The I/O independent runner executes a RAD instruction to obtain the extended result descriptor. It then marks the memory area containing the user program's buffer as available for movement by the memory manager module (MMG). The I/O independent runner then places the result descriptor into the system queue element that the queue R/D entry points to.

      The I/O independent runner clears the result descriptor field and the in process field of the queue R/D entry.

   d. The I/O independent runner releases (unlocks) the lock structure of the exchange.

      It then creates a "converted" result descriptor in the system queue element. This result descriptor is later posted to the user program's buffer status word.

   e. The I/O independent runner updates the I/O block count in the IOAT entry of the file. If necessary, it also performs routines to handle ISC and other special situations (depending on the Q-SPRT field).

f. The I/O independent runner finishes working with the system queue element. The actions performed by I/O independent runner at this point depend on what type of I/O was originally fired: "fire and wait" or "fire no wait". Remember that the LIO module *always* uses "fire no wait" I/Os.

Fire and Wait

On a fire and wait I/O, the task requesting the I/O is waiting until a successful I/O completes before it resumes processing. On this type of request, the I/O independent runner signals the event in the system queue element. It then returns to look for *additional* I/O's that may be waiting for the current exchange. (described under g. in the following paragraphs).

Fire No Wait

On a fire no wait I/O, the requesting task continues processing while the I/O is being processed. On this type of request, the I/O independent runner calls to a routine named PSTR/D (Post R/D).

The PSTR/D routine copies the appropriate portions of the system queue element back into the user queue element, including the "converted" result descriptor. It also copies the "converted" result descriptor from the user queue element into the buffer status word within the user program.

The PSTR/D routine signals the event in the user queue element.

The PSTR/D routine returns the system queue element to the linked list of available queue elements. It then begins to look for *additional* I/O's that may be waiting for the current exchange. (described under g. in the following paragraphs).

g. After handling one normal I/O complete, the I/O independent runner checks if there are any *additional* I/Os waiting for the current channel. The routine that handles additional I/Os is named DEQUE.

- The DEQUE routine locks the exchange lock.
- The DEQUE routine checks the queue R/D area to see if the channel is available. (Normally the channel is available).
- The DEQUE routine looks at the Exchange Head/Tail Table to see if any I/Os that can be fired on the current channel are waiting.

If there *is* an I/O waiting for the channel, the I/O is linked in process the channel, delinked from the "waiting" list of queue elements and initiated. (This process is described under 3 e. earlier in this section.) The I/O independent runner proceeds with the processing described under h. in the following paragraphs.

If there are *no* I/Os waiting for the channel, the I/O independent runner proceeds with the processing described under h. in the following paragraphs.

h. After handling any additional I/Os waiting for the current channel, the I/O independent runner releases (unlocks) the lock structure of the exchange. The I/O independent runner then checks for any additional *I/O completes* on other channels. These additional I/O completes may have occurred during the processing of the previous I/O complete or during the initialization of the additional waiting I/O.

The I/O independent runner checks the queue R/D area to see if any channels have generated an I/O complete.

- If there *is* a pending I/O complete, the I/O independent runner resumes the processing described beginning with step 6 of the preceding paragraphs.
- If there are no pending I/O completes, the I/O independent runner issues an interrupt instruction to call the kernel of the operating system. The I/O independent runner sleeps until it is next awakened by the kernel (when the next I/O complete interrupt occurs).

6. The user program that issued the READ BCT may be waiting on the event in the user queue element. After the I/O independent runner signals the event, the user program is placed in the ready to run list. At some point (based on the priorities of the tasks running on the system), the kernel reinstates the user program. When LIO determines that the event has been signalled, it returns control to the code of the user program, which can then operate on the data in its buffer.

SECTION 6
MAINTENANCE PROCESSOR

The maintenance processor (MP) is a separate processor within a V Series system that performs maintenance and diagnostic functions, including loading control store firmware into the system. The MP runs while the rest of the system is running, but many of its diagnostic functions require that the system be stopped.

This section lists some of the MP's capabilities that are useful for operating system support. It shows what to enter through the MP to obtain a desired result. It does not give detailed explanations of MP commands or information about how the MP works.

NOTE

This section describes how to use the maintenance processor on a Unisys V 300 system. This material only applies to V 300 systems.

To obtain definitive, detailed information about the maintenance processor, use the *V 300 Maintenance Users Guide* (commonly called the "MUG"). This document contains much more information about the MP and describes all of the MP's capabilities. Most of the information in this section is extracted from the *V 300 Maintenance Users Guide*.

This section is organized as follows:

- Using the MP. Includes a brief description of how to access the MP.
- Stop Capabilities of the MP. Includes the different types of MP stops and how to set up different MP stop conditions.
- Displaying or modifying system state. Includes how to display/modify the contents of memory, the accumulator and various data structures of the operating system.
- Other MP commands. Includes a small number of MP commands that can be useful for operating system support.

USING THE MAINTENANCE PROCESSOR

The maintenance processor (MP) is always running as long as the system is powered on. To enter commands through the MP for support purposes, however, use the following procedure:

1. Put the system into console mode.

   Console mode is a special system state where the ODT communicates with the maintenance processor. In this state, the ODT is called a maintenance console. There is a switch on the front of the system that changes the system's state from ON-LINE to CONSOLE.

   Press the ON-LINE/CONSOLE switch until the word CONSOLE lights up.

   The system may take a few moments before it changes mode. When the mode changes, the ODT display changes. The word RUNNING may appear.

   The ODT is now considered a maintenance console. You can now enter MP commands. At this point the system is still running.

2. Decide if the system must be stopped.

   To examine or change the system's state, the system must be stopped.

   To set up a stop condition (a condition that controls how and where the system will stop), the system can still be running.

3. Stop the system or enter maintenance processor commands.

   To stop the system, press the SPCFY key on the maintenance console. The system will stop executing instructions and display the word STOPPED on the console. At this point you can use any of several MP commands. Details about displaying or changing system state are included later in this section.

   To set up a stop condition, enter the characters MP and press XMT. The system can be running when this command is entered. MP is a maintenance processor command that places the maintenance processor in a state known (for historical reasons) as maintenance panel mode. A special screen is displayed on the console to let you define system stop conditions. Details about stop conditions are included later in this section.

STOP CAPABILITIES OF THE MAINTENANCE PROCESSOR

The maintenance processor can stop the operation of the system in several different ways, based on many different conditions. This ability can be used to examine the action of different pieces of software on the most detailed, hardware-based level.

This section summarizes the different ways that the system can be stopped and some of the stop conditions that can be defined through the maintenance processor. Read section 6.7, MP Stop Logic, of the *V 300 Maintenance Users Guide* (commonly called the "MUG") for more detailed information.

Types of Maintenance Processor Stops

The maintenance processor command MP is used to set up stop conditions for the system. After the MP command is entered, the maintenance processor displays a formatted screen used to enter stop conditions. One of the options on this screen controls the type of system stop. The following paragraphs summarize the stop types that are available through the maintenance processor. Some items in the stop type summaries may be unfamiliar. To understand a specific item, refer to the *V 300 Maintenance Users Guide*:

SYSTEM STOP

> Halts entire system within 2 clocks of the stop event. May be recoverable if IOP is not in data transfer. If I1DATA and I2DATA are reset then the stop is recoverable, otherwise there may be a non-recoverable I/O error because I/Os may be lost. Assumed by MP software to be non-recoverable. This stop type is not advised; it may cause media parity errors.

PROCESSOR STOP

> Halts processor (as opposed to the processor and the IOP) within 2 clocks of the stop event. May not be recoverable due to processor memory reads or writes being lost. Assumed by MP to be non-recoverable. Generally, this type of stop should not be used for operating system support.

INSTRUCTION STOP

> Halts processor (as opposed to the processor and the IOP) at the next single instruction breakpoint (after the Fetch phase of the next instruction in the Fetch module and after the rest of the pipeline has emptied). The IOP and memory continue to run unless a maintenance processor command is entered that accesses an IOP or memory chain; then IOP and memory are stopped when IOP IDLE active. Recoverable stop and most frequently used.

DRYUP

> Halts processor (as opposed to the processor and the IOP) within 8 clocks of the stop event and is completely recoverable by restarting the processor clock. Two clocks after the event, Memory R/W Control is sent the DRYUP signal to stop issuing read requests to memory. Six clocks later, all outstanding processor memory reads will be complete and the processor clocks are turned off. The IOP and memory continue to run unless the user requests a maintenance function that will access an IOP or memory chain; then, IOP and memory are stopped when IOP IDLE active. All memory writes are completed before the stop.
>
> Use this type of stop to stop the processor on conditions that may not occur at the same time as the SI signal (for example stopping on a memory write or stopping on an environment change).

Maintenance Processor Stop Logic

The maintenance processor can cause the operating system to stop execution at different points, based on different conditions. These conditions are referred to as stop logic.

To set up stop logic through the maintenance processor, the system must be in console mode. (This changes the ODT into a maintenance console.) The system does not have to be stopped before you enter stop logic. Enter the maintenance processor command MP. This command places the console in a state called (for historical reasons) maintenance panel mode. The maintenance processor displays a formatted screen.

The stop logic screen looks like this:

```
[*]  MP MODE
[ ] RZHERE  [ ] LDRBUS  [ ] WWSTA+  [ ] WMEMRQ  [ ] WOBTL    [ ] 1MSCLK  [ ]          [ ]
[ ] WADRER  [ ] USER    [ ] USER/   [ ] IOEXCP  [ ] IODRDY   [ ] RTIOCP  [ ] RQLNG+  [ ]
[ ] RADRER  [ ] OVF     [ ] TMOUT2  [ ] TMRINT  [ ] CPINT    [ ] W2BTER  [ ] RSTAT+  [ ]R2BTER
[ ] FADRER  [ ] NORML/  [ ] TMOUT1  [ ] 1BITER  [ ] RQCMD+   [ ]          [ ]          [ ]
[ ] ECSADR+
[ ] FCSADR+      [ ] PC+         [ ] OP+       [ ] RDRADR+
[ ] I1CSADR+     [ ] I1MLI ST+   [ ] I1JCNT+   [ ] I1ERROR      [ ] I1INTREQ     [ ] I1EMERG
[ ] I1CSADR+     [ ] I2MLI ST+   [ ] I2JCNT+   [ ] I2ERROR      [ ] I2INTREQ     [ ] I2EMERG
[ ] MEMWRITE+    [ ] MEMWRABS+   [ ] MEM BUS+  [ ] AWBUS+       [ ] RBUS+         [ ] FBUS+
[ ] BP535

[ ] SYSTEM STOP  [ ] PROC STOP  [ ]INSTR STOP  [ ] DRYUP   [ ] SNAP    [ ] CONTINUE

RSTAT: Enter *   HERE2[ ]   EMPTY[ ]   DUAL[ ]   SINGLE[ ]   OBTL[ ]
ECSADR = [  ]   WWSTA= [ ]  I1MLI ST= [ ]  BUS DATA= [        ]  OP= [    ]
FCSADR = [  ]   RQLNG= [ ]  I2MLI ST= [ ]  BUS CMD = [  ]         PC= [        ]
I1CSADR= [  ]  RQCMD=  [ ]  I1JCNT=   [ ]  BUS REQ = [ ]       RDRADR= [        ]
I2CSADR= [ ]                I2JCNT=  [ ]  BUS LEN = [ ]       MEMADR= [        ]
```

Figure 6-1. Maintenance Processor Stop Logic Screen

The following paragraphs show how to set up several frequently used stop conditions.

Stopping on a Particular OP Code

This condition stops execution when the Fetch module processes an instruction that contains a particular OP code. Enter the following information through the stop logic screen.

- Place a * next to OP+ on line 7.
- Place a * next to the desired stop on line 13 (Usually INSTR STOP).
- Fill in the two digit OP code on line 16.

Stopping on a Particular Instruction Address

This condition stops execution when the Fetch Module processes an instruction that starts at a particular address. Enter the following information through the stop logic screen.

- Place a * next to PC+ on line 7.
- Place a * next to INST STOP on line 13.
- Fill in the eight digit PC field on line 17. This field must include the base index digit (currently the base index digit is always 1).

  For example, if the address is in executable code, add 1,000,000 to the address. This will put a 1 in the high order digit of the PC field and tell the processor that the address is relative to Memory Area 1 (Code Memory Area). If the address was relative to Memory Area 2, 2,000,000 would be added, (currently the instruction addresses are always relative to memory area 1).

Stopping on a Memory Read

There are two different stop conditions that stop execution during a memory read operation. To understand the difference between the two, remember that a memory read operation goes through the following steps:

1. The Fetch or XM module determines that a memory read operation is needed. Then a read command is passed through the AW Bus to the memory control portion of the Memory Interface module to get the data.
2. Once the read command is executed by the memory control portion, it takes about 13 clocks to acquire the data from the Memory Interface module.
3. The data is made available to the Fetch or XM module through the Read Bus.

The Read Bus can deal with at most 10 digits of information. So for a large piece of data, a series of Bus commands may be issued to accomplish the job. If the data is less than 10 digits, only one command will be issued to get the data.

Stopping on Read AW Bus Command

This condition stops execution when a read command is passed to the memory control portion of the Memory Interface module. Enter the following information through the stop logic screen.

- Place a * next to AWBUS+ on line 10.
- Place a * next to INSTR STOP on line 13.
- Enter "05" in the BUS CMD field on line 17.
- Fill the BUS DATA on line 16 with the address of the data. Remember, the data is broken into 10 digits pieces. It is up to you to figure out the exact address that causes the processor to stop on the read.

Stopping on Read Bus Pattern

This condition stops execution when data matching a particular pattern is made available on the Read Bus. Enter the following information through the stop logic screen.

- Place a * next to RBUS+ on line 10.
- Place a * next to INSTR STOP on line 13.
- Fill out the BUS DATA field with the desired stop pattern. The BUS DATA field is a 10-digit field. When you enter a pattern for matching, enter the desired pattern in the significant digits and leave blanks for those digits that don't matter.

  For example, you could enter the following (where b represents a blank).

b02b44bbb0

The MP will match the pattern, digit by digit, for the data retrieved. Any value will be considered as matching the blank digits.

Stopping on a Memory Write

There are two different stop conditions that stop execution during a memory write operation. To understand the difference between the two, remember that a memory write operation goes through the following steps:

1. 1. First, the XM issues an AW Bus command WRITE ADDRESS (02), to the memory control portion of the Memory Interface module to indicate the address that XM intends to write to.
2. XM issues a series of WRITE DATA commands (0E) to the memory control portion (depending on the size of the data), to move the data to memory. The WRITE DATA command can hold at most 10 digits piece maximum. Subsequent commands concatenate data to the data written by the previous WRITE DATA command.
3. Write operations are concluded in two ways, based on whether the instruction requires only one logical write or requires multiple logical writes.
   a. For instructions that can be done in one logical write (such as MOVE instructions), the write operation is concluded by a WRITE PC OP COMPLETE AW Bus command (0B). This signals the end of the write operation. The address of the instruction that initiated the write is identified in the AW Bus command.
   b. For instructions that cannot be done in one logical write, (such as MOVE LINK instructions), several WRITE PC NOT OP COMPLETE AW Bus commands (0A) are included before the write operation is concluded. These commands instruct the memory control portion to start moving the incoming data into the new location. On the last logical write a WRITE PC OP COMPLETE (0B) command is issued to the memory control portion. The write operation is completed in the way described in the previous paragraph.

Stopping on a Write to a Particular Location

This condition stops execution when data is about to be written to a particular location (address).

The stop address can be specified through either the MEMWRITE+ or the MEMWRITE ABS+ fields on line 10. MEMWRITE+ accesses all operating system memory; MEMWRITE ABS+ accesses absolute memory (including operating system memory *and* the 10,000 digits of Sub-MCP Base memory). To access sub-MCP Base memory, use the MEMWRITE ABS+ field. To access normal operating system memory use either MEMWRITE ABS+ with 10,000 added to the address or use MEMWRITE+ (which adds the 10,000 for you).

Enter the following information through the stop logic screen.

- Place a * next to MEMWRITE ABS+ or MEMWRITE+ depending on your need.
- Place a * next to DRYUP on line 13.
- Fill in the stop address for MEMADR on line 19.

Stopping on Write AW Bus Command

This condition stops execution when a write command is passed to the memory control portion of the Memory Interface module. Enter the following information through the stop logic screen.

- Place a * next to AWBUS+ on line 10.

- Place a * next to INSTR STOP on line 13.
- Enter "02", "0E", "0A" or "0E" in the BUS CMD field on line 17.
- Fill the BUS DATA on line 16 with the address of the data. Remember, the data is broken into 10 digits pieces. It is up to you to figure out the exact address that causes the processor to stop on the write.

Stopping on an Environment Change

This condition stops execution when an instruction (VEN, HCL, BCT or RET) causes the system to change its memory environment. An environment change causes the processor to change the environment number stored in the Reader Scratch Pad (RSP) using the AW Bus command WRITE SCRATCH PAD (1A). This condition stops execution when the write command is passed to the memory control portion of the Memory Interface module. Enter the following information through the stop logic screen.

- Place a * next to AWBUS+ on line 10.
- Place a * next to DRYUP on line 13 (Do not use INST STOP!).
- Fill "1A" into BUS CMD on line 17.
- Fill "E" into BUS LEN on line 19.
- Fill BUS DATA on line 16 with the first six digits of the desired environment. Do not forget the "D" used as the base index digit for an MCP environment. For example, enter D00037bbbb for MCP environment number 37 (where b represents a blank space).

Stopping on an IIO to a Particular Channel

This condition stops execution when an IIO instruction is issued for a particular channel. To stop execution when *any* IIO instruction is issued, use the procedure shown in Stopping on a Particular OP Code, earlier in this section.

The Fetch module retrieves an instruction from memory, parses it and passes it through FBUS to one of the two Fetch pages. The XM executes the instructions in the Fetch pages, one at a time. This stop condition stops on the write to a fetch page on FBUS. Enter the following information through the stop logic screen.

- Place a * next to FBUS+ on line 10.
- Place a * next to INSTR STOP.
- Set BUS LEN to 1. (Here the BUS LEN is the location of the word in the Scratch Pad.)
- Fill in BUS DATA according to the format of word 1 of the fetch page. For example, enter 94bb08 (where b represents a blank space) to stop the processor when an IIO is fired on channel 08.

Stopping on an Error Condition

This condition stops execution on an error condition, before control passes to the Operating system's fault handler routine. This stop condition is *not* set up through the stop logic screen. Enter the following commands when the system is in console mode and execution is stopped.

CAUTION

Use this procedure at your own risk. Do not use this procedure unless absolutely necessary.

This procedure changes the parity of a specific piece of memory to freeze processing and prevent the operating system from invoking the fault handler. This procedure has inherent risks and should only be performed when absolutely necessary. It should always be performed under controlled, supervised conditions.

After this procedure is performed, you will be unable to DS any programs. The operating system will be unable to respond to any soft fault conditions. All stack frame information must be displayed manually, through the MP. If you perform the procedure incorrectly, the resulting problems are extremely difficult, if not impossible, to interpret. When the operating system's fault handling capabilities are disabled, the *results are unpredictable.*

The addresses and data shown in this procedure are subject to change as operating system development progresses. New releases of system micro-code (also known as firmware) frequently change the documented addresses and data.

Enter the following maintenance processor command:

```
ECRAM@2441
```

A 20-digit value will be displayed. For example:

```
ECRAM@2441 = 0D00 0000 A808 0000 01F0
```

Reverse any one bit of the first digit and transmit all 20 digits at once. For example:

```
ECRAM@2441 = 8D00 0000 A808 0000 01F0
```

This causes a SNAP to be taken when a fault or error happens. Examine the SNAP picture to find out what is causing the problem.

To reinstate the fault handler without having a SNAP taken, change the bit in ECRAM back to its original value. To be certain that normal operation can resume, reload the control store firmware.

An ESCADR+ stop at address 2441 (using processor stop or a dryup stop, but not an instruction stop) can sometimes be used instead of the ECRAM command. This type of stop may be recoverable. When the processor is stopped using this procedure, the processor R/D causing the fault is available using an ESP@1C command.

DISPLAYING AND MODIFYING SYSTEM STATE

The following pages summarize several ways to display or modify the state of the system through the maintenance processor (MP). The MP command syntax and possible result values are provided. For details and reasons, refer to the *V 300 Maintenance Users Guide* (commonly called the "MUG") or the *V 300 System Reference Manual.*

The MP commands on the following pages are used when the system is in console mode and after the system is stopped. See Using the Maintenance Processor earlier in this section for general information about console mode and stopping the system. Some of the commands and some options require that the system debugging flexible disk named PANSMV be loaded. These commands and options are noted.

Display/Modify Contents of Memory

To read or write memory at a given absolute or operating system base address, use the MEM command.

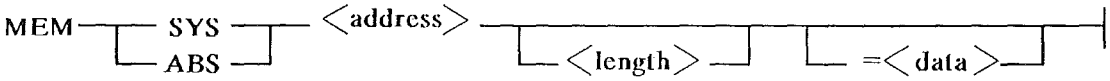


Figure 6-2. MEM Command (Format 1)

ABS gives an absolute address. SYS gives an operating system base relative address. The default length for display is 50 digits.

To read memory at a given V Series virtual address without altering the state of the processor, use the following syntax available only on the PANSMV minidisk. Detailed information about this command is included in section 8.13 of the *V 300 Maintenance Users Guide.*

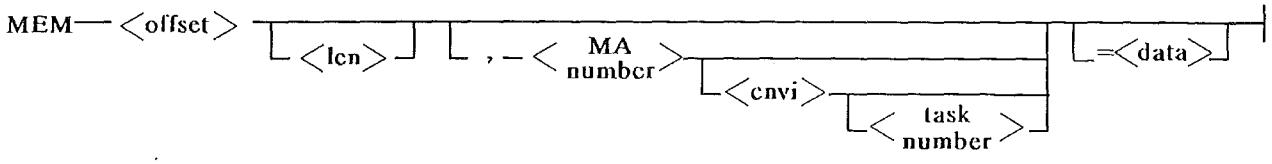


Figure 6-3. MEM Command (Format 2)

Display/Modify Accumulator

To display or modify the accumulator of the currently executing task, enter:

ESP@1D 3

To display or modify the accumulator of any other task on the system, the PANSMV flexible disk must be loaded. Enter:

RLE <task no> 114 28

This command lists the reinstate list entry of the task with the entered <task no>. The accumulator is located at offset 114 from the base of the Reinstate List Entry.

Display Base/Limit Register

The eight base limit pairs of the executing task are located in the Base/Limit table of the Memory Interface module, stored in the Reader Scratch Pad (RSP). To display the desired base/limit pair, enter:

RSP@n (where 0 <= n <= 7)

The value displayed is in the form BBBBBLLLLL where BBBBB and LLLLL are the base and limit addresses of the memory area, truncated to an even 1,000 digits.

To display base/limit pairs of other tasks, use the task's reinstate list entry to obtain the task's environment table (ET) address, display the ET and obtain the active memory area table (MAT) address. Display the active memory area table.

Display a Task's Stack Frame

A stack frame of a individual task can be displayed on the maintenance console. The PANSMV flexible disk must be loaded. To display the stack frames of an individual task, use the STACK and NSTACK commands.

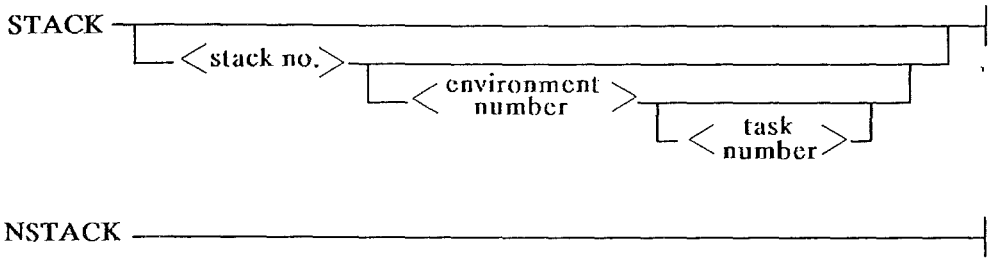


Figure 6-4. STACK and NSTACK Commands

STACK

The default syntax of the STACK command displays the topmost stack frame within the current environment of the current task.

A stack number of 0 refers to the topmost stack frame within the environment of the task. A stack number of 1 refers to the next frame down in the stack, and so on.

NSTACK

NSTACK displays the next stack frame down in the stack determined by a previously entered STACK command. This command cannot be entered unless preceded by a STACK command or another NSTACK command.

Display Date and Time

To display the Time-of-day Timer located in Sub-MCP Base Area, enter:

MEM ABS 340 20

The date and time are displayed as YYYYMMDD00SSSSSSSS000 where YYYY is the year, MM is the month, DD is the date and SSSSSSSS is milliseconds since midnight.

Display Environment Table Entry Address

To display or modify the address of the current task's ET entry, enter:

RSP@B (for operating system's ET address)

RSP@C (for task's user ET address)

The format is 00AAAAAAAA, Where AAAAAAAA is the absolute address. Optionally, ET entry address can be resolved from the task's reinstate list entry. (This capability is only available when the PANSMV flexible disk is loaded.) Enter:

RLE <task no> 20 8

Display/Modify Index Register

The index registers of the current task can be displayed or modified as a group or individually. To display all seven index registers of the current task enter:

FSP@9 7

This syntax assumes that IX1, IX2 and IX3 have been used. They are valid only if they *have* been used. In many situations, this assumption is true.

To display a single index register of the current task, enter:

FSP@<n> (n = 8 + number of desired register)

For tasks other than the current task, display IX1, IX2, IX3 in Memory Area 0 of the current task's memory environment. IX4, IX5, IX6, IX7 are stored in the task's reinstate list entry.

Display/Modify Interrupt Mask

To display the interrupt mask of the current task, enter:

RSP@E

The format of the displayed mask is IIIIIIHHMM, where IIIIII is the active environment number, HH is two digits of interrupt history, and MM is the two-digit interrupt mask.

Table 6-1. Interrupt Mask Layout

Condition	Bit	Cause
Reserved	7	
Instruction	6	Instruction-related interrupt
Overtemp	5	System over-temperature
Task Timer	4	MSD = 0
Reserved	3	
REAL TIME I/O	2	Real time I/O complete, no exceptions

Table 6-1. Interrupt Mask Layout (Continued)

I/O ERROR	1	I/O complete, exceptions
NORMAL I/O	0	Non-real time I/O complete, no exceptions

Display Interrupt History

To display the Interrupt history, enter:

```
RSP@E
```

The maintenance processor displays IIIIIIHHMM where HH is the Interrupt history and MM is Interrupt Mask.

Display/Modify Pointer To Memory Area Status Table

To display the system's pointer to the memory area status table (MAST), enter:

```
MEM ABS 222 8
```

Display Memory Area Table Entry

A memory area table (MAT) entry can only be displayed if the PANSMV flexible disk is loaded. The maintenance processor command MATE is used. The syntax of the MATE command is as follows:

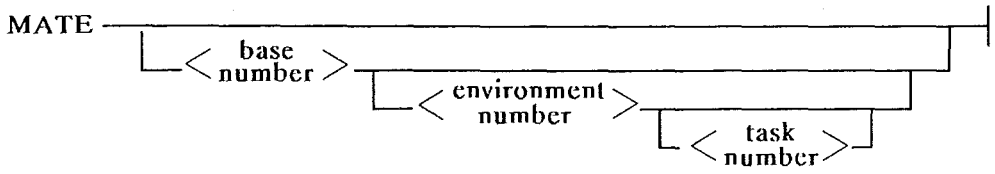


Figure 6-5. MATE Command (Maintenance Processor)

The value displayed is 20 digits. If you enter:

```
MATE 1 U 25
```

The maintenance processor displays:

```
MEM ABS 00098765 = C000000001 0000000000
```

The information displayed by the MP is broken down differently depending on the type of entry in the memory area table. If the entry is an original entry, the layout is as follows:

Table 6-2. Memory Area Table (Original Entry)

Digits	Information
0:5	Base Address
5:5	Limit Address
10:4	Memory Area Status Table number
14:6	Software Use

If the entry is a copy entry, the layout is as follows:

Table 6-3. Memory Area Table (Copy Entry)

Digits	Information
0:1	Type = "C"
1:1	Reserved
2:6	Environment Number
8:2	Memory Area Number
10:10	Software Use

If the entry is a memory area fault entry, the layout is as follows:

Table 6-4. Memory Area Table (Memory Area Fault Entry)

Digits	Information
0:1	Type="F"
1:1	Reserved
2:8	Faulted Area Table Address
10:10	Software Use

If the entry is an unused entry, the layout is as follows:

Table 6-5. Memory Area Table (Unused Entry)

Digits	Information
0:1	Type = "B"
1:9	Reserved (must be zero)
10:10	Software Use

Display Memory Area Table

A memory area table (MAT) can only be displayed if the PANSMV flexible disk is loaded. The maintenance processor command MATL is used. The syntax of the MATL command is as follows:

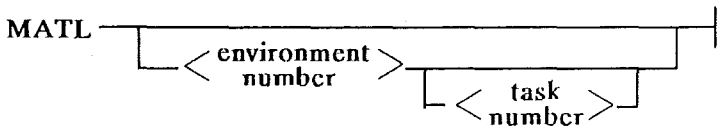


Figure 6-6. MATL Command (Maintenance Processor)

If you enter:

```
MATL U1 4
```

The maintenance processor displays the first eight entries of the specified MAT:

```
MEM ABS 00004000 - C000000001 0000000000 C000000001
                                0000000000 B000000000
MEM ABS 00004050 - B000000000 B000000000 B000000000
                                B000000000 B000000000
MEM ABS 00004100 - B000000000 B000000000 B000000000
                                B000000000 B000000000
MEM ABS 00004150 - B000000000
```

Figure 6-7. MATL Command (Maintenance Processor Output)

Display/Modify Task Timer

A task timer is a counter that is used to interrupt a task when its time slice has ended. To display the current task's task timer, enter:

```
MEM ABS 360 6
```

To display the time slice remaining for a task, the PANSMV flexible disk must be loaded. Enter:

```
RLE <task no> 108 6
```

To display the new time slice of a task, the PANSMV flexible disk must be loaded. Enter:

```
RLE <task no> 76 6
```

Display/Modify Comparison Toggles

To display the comparison toggles of tasks other than the currently executing task, the PANSMV flexible disk must be loaded. Enter:

RLE <task no> 184 2

Display/Modify Mode Toggles

The mode toggles reflect the operating mode of the system. For more detailed information, refer to section 4.3.5 of the *V 300 Maintenance Users Guide*. To display the current task's mode toggle, enter:

RFF

The maintenance processor displays a ten-digit value. The mode toggles are the last two digits. When the mode toggles are displayed using this syntax, they are formatted differently than when they are displayed as part of a reinstate list entry. The mode toggles of tasks not currently executing are included in the respective tasks' reinstate list entries.

Table 6-6. Mode Toggle Values

Bits	Information
8	SNAP Enable
4	Soft Fault Enable
2	SNAP Mem avail
1	Trace

Display/Modify Measurement Register

To display or modify the measurement register of the currently executing task, enter:

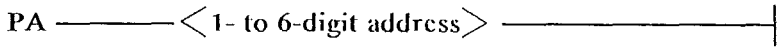
RSP@F

The first 8 digits are the measurement register. For tasks other than the current task, the PANSMV flexible disk must be loaded. Enter:

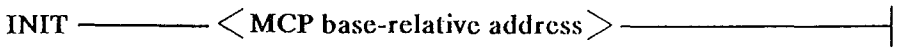
RLE <task num> 142 8

Display/Modify Program Address Register

The current value of Program Address Register is displayed on the maintenance console. The maintenance processor command PA can be used to alter the instruction flow within the same memory environment. This procedure is described in section 8.16 of the *V 300 Maintenance Users Guide*.



The maintenance processor command INIT can be used to change to a different memory environment. This command is not recommended for use. INIT initializes the system from the MCP base-relative memory address indicated in the command. Before using the INIT command, read section 8.8 of the *V 300 Maintenance Users Guide*.



Display Interrupt Cause Descriptor

To display the Interrupt Cause Descriptor, enter the following maintenance processor command. The possible interrupt descriptor values are shown in the following table.

```
MEM SYS 32 2
```

Table 6-7. Interrupt Descriptor Values

Value	Cause
08-FF	Reserved
07	Failed Virtual Branch Reinstate
06	Instruction Interrupt
05	Failed Hardware call
04	Released Event
03	Released Lock
02	Failed Event
01	Failed Lock
00	Reserved

Display State Toggle

To display the State Toggle in the read control portion of the Memory Interface Module, enter the following maintenance processor command. Information about the Read Flip Flop (RFF) bit assignment is contained in the *V 300 Maintenance Users Guide*.

```
RFF
```

Display Reinstate List Entry

The maintenance processor command RLE is used to display reinstate list entries. The PANSMV flexible disk must be loaded to use the RLE command. The syntax of the RLE command is as follows:

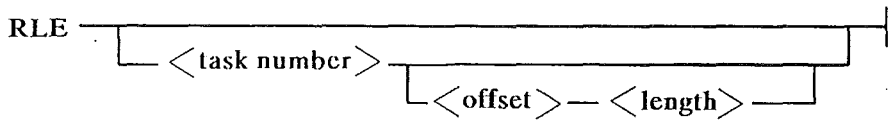


Figure 6-8. RLE Command (Maintenance Processor)

Display Reinstate List Pointer

To display the reinstate list pointer, enter the following command. When the maintenance processor displays a response, subtract 10,000 to obtain the MCP base-relative address.

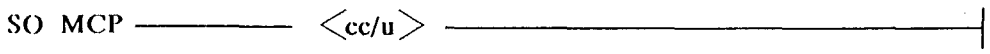
```
MEM ABS 240 10
```

OTHER MAINTENANCE PROCESSOR COMMANDS

The following pages contain syntax diagrams summarizing the syntax of several maintenance processor (MP) commands. These diagrams are only provided for quick reference, full explanations of the commands are included in the V 300 Maintenance Users Guide. The commands are labelled according to their use.

Set MCP Disk Channel and Unit

The maintenance processor command SO sets system options. One of the most frequently adjusted system options is the channel and unit of the device where the operating system code file resides. Detailed information about the SO command is provided in section 8.27 of the *V 300 Maintenance Users Guide*. The syntax of the SO command used to set the operating system channel and unit is as follows:



Load File From Flexible Disk (Minidisk)

The maintenance processor command LOAD has several options. For details see section 8.9 of the *V 300 Maintenance Users Guide*. The syntax of the LOAD command is as follows:

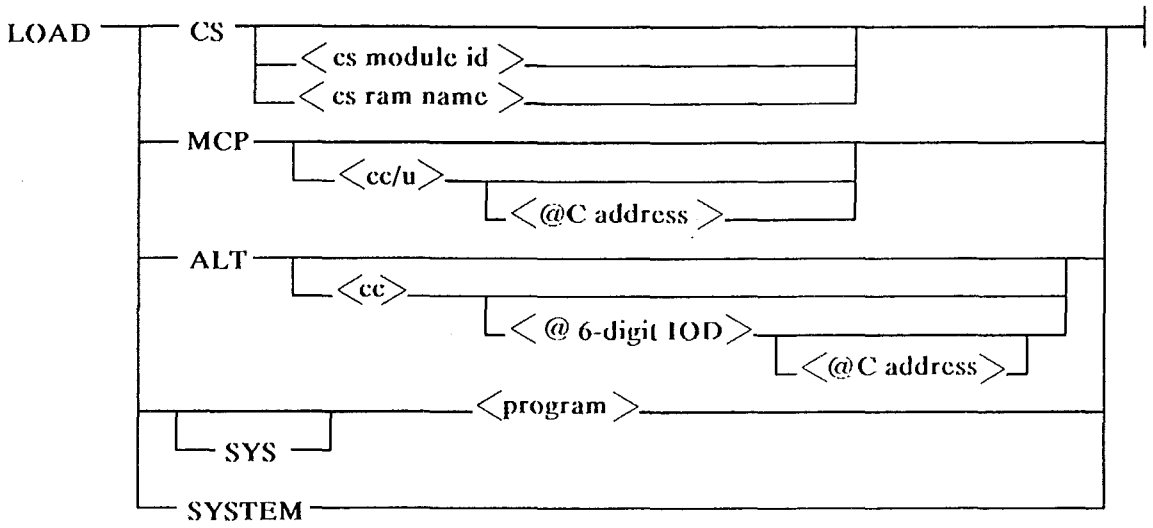


Figure 6-9. LOAD Command (Maintenance Processor)

Setting System Options

The maintenance processor command SO sets several system options. For more details see section 8.27 of the *V 300 Maintenance Users Guide*. The syntax of the SO command is as follows:

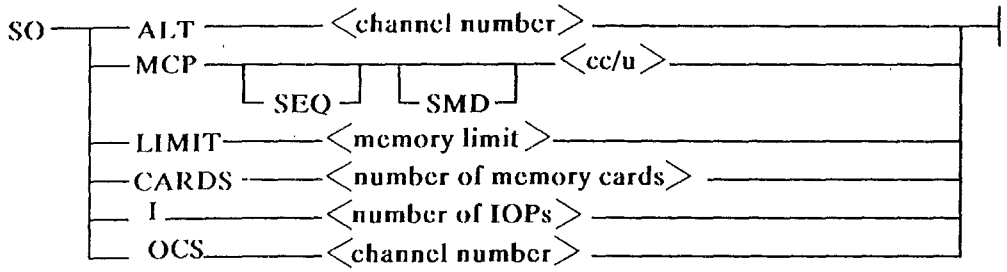


Figure 6-10. SO Command (Maintenance Processor)

Displaying System Options

The maintenance processor command TO sets several system options. For more details see section 8.32 of the *V 300 Maintenance Users Guide*. The syntax of the TO command is as follows:

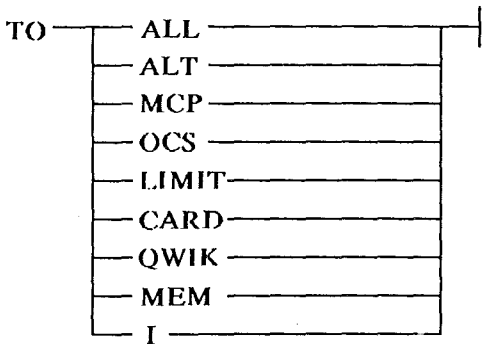


Figure 6-11. TO Command (Maintenance Processor)

SECTION 7
TRACE

The fault handler module (FAU) of the MCP/VS operating system contains a built-in instruction trace facility.

CAUTION

Use this trace facility at your own risk! This facility is designed for internal use during the maintenance of the MCP/VS operating system. It is *not* a supported feature. It should never be used on a production system. It should only be used by experienced operating system support personnel who are well aware of its capabilities and dangers.

This trace facility currently contains several known software errors. It may contain additional unknown software errors. Reported software errors will not be corrected.

This trace facility can distort the timing between and within various software routines. Such timing distortion is a common characteristic of all procedure traces. In some cases timing distortion prevents the original problem being examined from appearing during the trace. In some cases timing distortion creates additional software errors that do not occur during normal operation.

The trace facility within the fault handler module (FAU) is a low-level instruction trace that supplements the interactive DEBUG module. The interactive DEBUG module cannot trace the physical I/O module (IOM); the low-level instruction trace is able to trace IOM. The low-level trace *cannot* trace the kernel module (KER) of the operating system.

The low-level trace is designed for debugging the I/O path of the operating system. It can also be used in any situation where you need to know exactly what is happening in other portions of the operating system or in user programs. It is often used to debug situations where several tasks interact to produce a given result. The low-level instruction trace is similar in many ways to the trace produced by the DBUG module of the MCPIX operating system (predecessor to MCP/VS).

TRACE PROCEDURE

To use the low-level trace facility within the fault handler module, follow these steps.

1. Make sure that there is a printer available on the system and that it is ready. The printer must be able to print 132 characters on each line and be able to print lower case letters. (This requires at least a 64 character train. A 96 character train is preferred.)

2. At the ODT, set up a stop condition using the MP command of the maintenance processor. Do *not* set up any of the following stop conditions.

   - A stop condition within the kernel module (KER) of the operating system. The low level trace facility cannot trace the kernel.
   - A stop condition involving an instruction that changes the memory environment of the system (examples include a BCT or a VEN instruction). If you need to stop on such an instruction, you must remember to single instruct *past* the environment change *before* you start the low-level trace.

   A sample stop condition is shown later in this section. Details about stop conditions are provided in section 6, Maintenance Processor.

3. The system stops when the stop condition is satisfied. After the system is stopped:

   - Use the MEM SYS command of the maintenance processor to set up trace parameters starting at memory address 3500. Be certain to at least set the first digit to a non-zero value (Example: MEM SYS 3500 = 3). Trace parameters are described later in this section and an example of a trace set up is also included.
   - Examine the full set of trace parameters. When any previous traces are ended, their parameters remain in place even though the trace is finished. Make sure that the trace parameters set up the type of trace that you want. Set any undesired trace parameters to their desired values.
   - Remember that the starting memory address of 3500 applies to a V 300 system and is dependent on the release of the microcode (sometimes called "firmware"). Different releases of microcode may require a different memory address.

4. Enter the maintenance processor command RFF. Note the response displayed on the screen. Change the low-order bit of the response and write the changed value back into the system's memory. For example, if the response displayed is

   000000005A

   You would enter the following command. (Notice that leading zeros are not required).

   RFF = 5B

   Do not forget to execute this step. This actually triggers the trace facility.

5. Start the processor by pressing the SPCFY key or by entering RUN and pressing XMT. Tracing will begin.

   - You can stop the printer to examine part of the trace and then ready the printer again. You will not lose any trace data by doing this. If you are tracing any set of tasks other than all tasks, stopping the printer drastically changes the timing of the system.
   - During a trace, the system may "lock up" and run at trace speed, even if only one task is being traced. This happens when the task being traced owns lock structures that are needed by other tasks (for example, the OCS independent runners).

6. To end the trace, stop the processor using the maintenance processor (details are provided in section 6, Maintenance Processor). After the processor is stopped, use the MEM SYS command to turn off the trace flag by entering:

   ```
   MEM SYS 3500 = 0
   ```

   It is helpful to set all other trace parameters back to their original values, but it is not required. (Memory address 3500 applies to a V 300 system and is dependent on the release level of the microcode — sometimes called "firmware". Different releases of microcode may require a different memory address.)

TRACE PARAMETERS

The following table shows the trace parameters as they are laid out starting at memory address 3500. The starting memory address of 3500 applies to a V 300 system and is dependent on the release of the microcode (sometimes called "firmware"). Different releases of microcode may require a different memory address.

Table 7-1. Trace Parameters for Low-level Instruction Trace

Memory Address	Parameter Description
3500	Trace Mode Flag
bit 8	IIO and VEN/RET Trace
bit 4	Trace to Tape (not currently implemented)
bit 2	Trace User Mode Operation
bit 1	Trace MCP Mode Operation (This digit is most commonly set to 3)
3501	Additional Trace Mode Flags (not currently implemented)
3502 to 3503	Channel Number of Printer (or Tape) Used when tracing the operating systemduring coldstart, halt/load, or any timebefore the IOATs are built. Also used ifprinter is not ready when system is stopped.
3504	Tape Unit Number (not currently implemented)
3506 to 3511	Printer Op Code Train Printer = 42140E Buffered Printer = 42100E Used when tracing the operating system during coldstart, halt/load, or any time before the IOATs are built. Also used if printer is not ready when system is stopped.
3512 to 3599	Array of 4-digit Task Numbers of Tasks to Trace (To trace *all* tasks, the first number must be "AAAA")
3552 to 3647	Array of 6-digit Environment Numbers of Environments to *NOT* Trace (By default, the first environment number in this array is initialized to D00002 — currently IOM.)
3600 to 3647	Array of 6-digit Environment Numbers of Environments to Print (To trace all environments, the first number must be "CCCCCC")

TRACE EXAMPLE

The following paragraphs show an example of trace set up using the low-level trace facility in the fault handler module (FAU).

1. Place the system in console mode. Details are provided in section 6, Maintenance Processor.

2. Enter the maintenance processor command MP. The stop logic screen is displayed. Set up a stop condition.

   This example shows a stop condition that causes the system to stop on any entry to or return to the operating system's environment number D00002, which is currently the physical I/O module (IOM).

   - Place a * next to AWBUS+ on line 10 of the screen.
   - Place a * next to DRYUP on line 13.
   - Enter D00002 in the BUS DATA field on line 16.
   - Enter 1A in the BUS CMD field on line 17.
   - Enter E in the BUS LEN field on line 19.

3. After the system stops:

   - Enter MEM SYS 82 4. This displays the current task number. In this example, assume the task number is 102.
   - Enter MEM SYS 3500 100. When the response is displayed, change the first digit to a 3. Change the first group of "AAAA" to the desired task number (in this case 102). Because this example traces IOM, you must override the default value inserted at address 3552. (The default sets IOM as an environment to *not* trace. Change the first group of "D00002" to "CCCCCC". When finished, you should have

     3000004214 0E0102BBBB AAAABBBBAA AABBBBAAAA ...
     BBCCCCCC DDDDDDCC ...

     This traces task 102 starting at the entry point into the IOM module, in both user and MCP mode.

   - Enter RFF. This displays the current RFF register. In this example assume the value displayed is 000000005A.

     Enter RFF = 5B. This sets the low-order bit of the RFF field.

4. Start the system. To do this, press the SPCFY key or type RUN and press the XMT key.

   Tracing will begin.

SECTION 8
TRAK

TRAK is a real-time debugging tool available under the MCP/VS operating system. TRAK works while the system is running. It records various data structures of interest to operating system support personnel. By analyzing these structures, you can follow a sequence of events within the operating system or judge if a module is working correctly.

TRAK consists of special "calls" embedded at strategic points within the operating system. These calls are enabled only when the TRAK option is set. When the option is set, the TRAK module of the operating system saves the data requested in the parameters of the call. The saved data is stored in a circular buffer called the TRAK buffer (sometimes TRAKBUFFER). This buffer can be formatted and printed by the system memory dump analysis program DMPANL. The size of the buffer is controlled by the LIMIT TRAKBUFFER record in the system configuration file.

After they are enabled by the TRAK option, TRAK calls are activated using syntax provided by the BT command. They can be activated inclusively (all TRAK codes), selectively by module (using the module's TRAK name), or individually (by 4-digit numeric code). The syntax of the BT and ET commands are described later in this section. SPRASM names of individual modules are included in table 1-4 in section 1, Operating System Modules.

A large number of TRAK calls are embedded in the operating system when it is released for distribution. These calls are inserted during development. Information about a limited subset of these calls is included in the DMPANL program. When DMPANL prints the TRAK buffer, it can include brief descriptions of these TRAK calls.

TRAK calls can also be patched into the operating system by operating support personnel. Information about a specific problem can be obtained by including TRAK calls at strategic points within a module or modules. The DMPANL program cannot provide descriptions of TRAK calls inserted outside of the development environment.

CAUTION

Patching a TRAK call into the operating system is a delicate procedure. It should *only* be attempted by experienced operating system support personnel.

The results of an incorrectly-patched TRAK call are unpredictable at best, and at worst can lead to problems that are totally unrelated to the original problem. This type of second generation" error is extremely difficult (if not impossible) to find and correct.

Be *extremely* careful when inserting a TRAK call into the operating system.

Remember the following points if you decide to insert a TRAK call in the operating system.

- TRAK calls should be located where a certain result is expected. This allows the actual run-time value to be compared with the predicted result for verification.
- TRAK calls should not be located within loops unless absolutely necessary.
- TRAK calls can also provide an audit trail; they can record the events that occur as a task progresses through a module or modules.

TRAK PROCESS SUMMARY

Here is a step-by-step summary of the way TRAK is used.

1. In many situations, TRAK calls already embedded in the operating system are sufficient to diagnose a problem. If the embedded TRAK calls do not apply to the particular situation, insert TRAK calls into the operating system as follows.

   ASSEMBLER

   ```
   LIX     DOVRD7,IXn
   VEN     PARMS,TRAUDT+Xn
   ```

   Where n represents the number of an index register from 1 to 7

   SPRITE

   track.enter @ (trak_ code, trak_ options, trak_ data)

   Track.enter and TRAUDT are the SPRITE and ASSEMBLER names of the same variable. This variable is a PTR TO PROC and is located in the System Tables memory area in a data block named trak_ code. When the TRAK option is not set, TRAUDT contains an environment pointer to a global return. This causes all TRAK calls to return to the caller. When the option is set, TRAUDT points to the main entry point in the TRAK module (named TRAK).

   TRAK parameters (PARMS, trak_ code, trak_ options, trak_ data) are described later in this section.

   If the TRAK call is located in a time-critical path of the operating system, you must first test the TRAK module in use flag before making the call. In SPRITE the flag is named hl.track_ options.module_ in_ use, in ASSEMBLER the flag is bit 4 of TRAK-F. The TRAK call should be made only when this flag is set.

2. Enable the TRAK option by entering SO TRAK at any ODT. This step will only work if the TRAK option is not already in use.

3. Activate TRAK calls by entering the keyboard command BT at the ODT. The full syntax of this command is as follows:

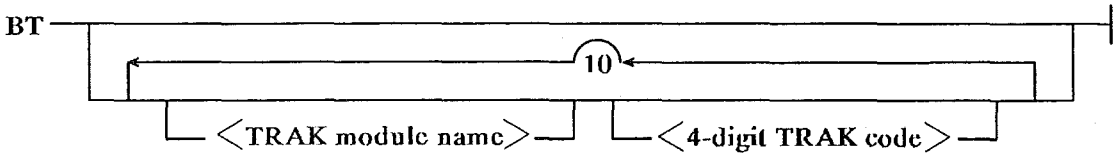


Figure 8-1. BT Command (Begin TRAK)

The BT command lets you activate all TRAK calls, TRAK calls within a maximum of ten selected modules (by module TRAK name), or a maximum of ten individual TRAK calls (by 4-digit TRAK code). TRAK names (SPRASM names) of individual modules are included in table 1-4 in section 1, Operating System Modules.

Examples

BT	(Activates all TRAK calls)
BT DCP, DMS	(Activates all TRAK calls within the DCP and DMSII modules)
BT 9700, 9744	(Activates two individual TRAK calls, using 4-digit codes)

If the previous TRAK session was ended with an ET command, the BT command erases the previous contents of the TRAK buffer.

If a TRAK session is currently in progress, you can enter a second BT command. This changes the TRAK specifications (TRAK specifications control which TRAK codes are placed in the TRAK buffer). When the second BT command is entered, its specifications *replace* the specifications entered through the first BT command. The data from the second BT command is inserted in the TRAK buffer after the last TRAK entry from the previous command. Data from the previous command remains in the TRAK buffer until the buffer's size is exceeded and the operating system begins overwriting the beginning of the buffer.

4. After TRAK calls are activated, information is inserted in the TRAK buffer. To deactivate TRAK calls, enter the ET command. This command has the following syntax:

Figure 8-2. ET Command (End TRAK)

This command deactivates all TRAK calls that were activated by the previous BT command. This command must be entered before the TRAK buffer is printed by DMPANL (using the 0 DM command). If the ET command is not entered, DMPANL will fail when it attempts to print out the TRAK buffer.

5. To print the TRAK buffer, create a memory dump file using the command 0 DM. Then print the memory dump file using the command PM 1 NOTBL TRAK.

6. When finished, remember to reset the TRAK option using the RO TRAK command. System performance may be slightly degraded if the TRAK option is not reset.

This sequence of events completes an entire TRAK session. You should always enter these commands in the order described. Any other combinations produce error messages.

TRAK PARAMETERS

The parameters of a TRAK call are described in the following paragraphs.

- Every TRAK call automatically inserts a default TRAK header into the TRAK buffer. This header contains:
  - calling module's environment
  - next instruction address
  - time stamp
  - all index registers
  - values of the TRAK parameters passed on the TRAK call

  Additional information is written into the TRAK buffer depending on the parameters passed.
- The TRAK parameters passed with an individual TRAK call are as follows:

Table 8-1. TRAK Parameters

Param. Name	Param. Size - Type	Description
T-CODE	4 UN	A 4-digit code used to identify the tracked information. Some codes are reserved for TRAK codes inserted during development. In SPRITE, this parameter is type HEX4.
T-OPTN	2 UN	Two digits that describe data structures to be tracked in addition to the default header. The first digit is available. The second digit is defined as follows: BIT8 : trace of last ten stack entries (environment and address only) BIT4 : last stack entry's stack parameters BIT2 : Q1AREA BIT1 : Q2AREA In SPRITE, this parameter is type TRAK_ OPTIONS.
T-ADDR	20 UN	The address of **any** data in memory that is placed into the TRAK buffer. This parameter is further divided into: 6 UN Environment number 2 UN Memory area number 6 UN Offset 6 UN Length In SPRITE, this parameter is type TRAK_ PARMS.

PARTIAL TRAK CODE LIST

The following pages contain a partial list of the TRAK codes inserted in the operating system during development. This list is incomplete and particularly subject to change. TRAK codes are added and deleted as the need for them arises. For definitive information about TRAK codes, refer to the source listing of the operating system.

Message Module TRAK Codes (1200 Series)

1201

broadcast

1202

converse

1203

continue

1204

end

1205

interrogate.

1206

int_ async.

1207

int_ one.

1208

int_ par.

1209

notify.

1210

position.

1211

receive.

1212

rcv_ one.

1213

rcv_ mine.

```
1214
        rcv_ par.
1215
        remove.
1216
        reply.
1217
        resend.
1218
        resendsn .
1219
        send.
1220
        sendsn.
1221
        send_ async.
1222
        signal.
1223
        signalp.
1224
        speak.
1225
        talk.
1226
        srv_ async.
1227
        srv_ msg.
1228
        ret_ node.
```

1229

ret_ message.

1230

ret_ message_ hdr.

1231

status_ interrogate.

DMS TRAK Codes (8200 Series)

8201

trak_ io_ rebuilt_ wait. Includes the DMSII queue element, Q1 queue element and Q2 queue element in TRAK buffer. Call occurs just after the DMSII queue element has been created and just before the DMSII module calls io_ mod to initiate a DMS wait type I/O. The DMSII queue element has been rebuilt from a pool of used DMSII queue elements.

8202

trak_ io_ rebuilt_ wait_ ioc. Includes the DMSII queue element, Q1 queue element and Q2 queue element in TRAK buffer. Call occurs just before control passes to one of the DMSII "handle_ ioc" routines. This routine processes I/O completes of all DMS wait type I/Os that use a rebuilt DMSII queue element (the type of I/Os that generate a TRAK code 8201).

8203

trak_ io_ rebuilt_ no_ wait. Includes the DMSII queue element, Q1 queue element and Q2 queue element in TRAK buffer. Call occurs just after the DMSII queue element has been created and just before the DMSII module calls io_ mod to initiate a DMS non-wait type I/O (also referred to as a "multi-threaded" I/O). The DMSII queue element has been rebuilt from a pool of used DMSII queue elements.

8204

trak_ io_ wait_ io. Includes the DMSII queue element, Q1 queue element and Q2 queue element in TRAK buffer. Call occurs just after the DMSII queue element has been created and just before the DMSII module calls io_ mod to initiate a DMS wait type I/O. This TRAK code is similar to TRAK code 8201, except that the DMSII queue element has not been rebuilt from a pool of used DMSII queue elements.

8205

trak_ io_ wait_ ioc. Includes the DMSII queue element, Q1 queue element and Q2 queue element in TRAK buffer. Call occurs just before control passes to one of the DMSII "handle_ ioc" routines. This routine processes I/O completes of DMS wait type I/Os that do not use a rebuilt DMSII queue element (the type of I/Os that generate a TRAK code 8204).

8206

trak_ io_ no_ wait_ io. Includes the DMSII queue element, Q1 queue element and Q2 queue element in TRAK buffer. Call occurs just after the DMSII queue element has been created and just before the DMSII module calls io_ mod to initiate a DMS non-wait type I/O (also referred to as a "multi-threaded" I/O). This TRAK code is similar to TRAK code 8203, except that the DMSII queue element has not been rebuilt from a pool of used DMSII queue elements.

8207

trak_ io_ enter_ handle_ ioc. Includes the DMSII queue element, Q1 queue element and Q2 queue element in TRAK buffer. Call occurs just before control passes to any of the DMSII "handle_ ioc" routines, regardless of any other TRAK calls.

ISC TRAK Codes (8500 Series)

8500

start_ of_ isc_ open. The operating system has started to open an ISC file.

8502

set_ amr_ result. The operating system has written information to the AMR (for the purpose of opening or closing an ISC file).

8504

read_ hub_ index_ result. An error has happened on a Read Hub Index operation while the operating system is trying to open an ISC file. The operating system cannot successfully communicate with the hub.

8506

device_ state_ unit_ bypass. The ISC is available when the operating system attempts to open an ISC file.

8508

device_ state_ unit_ assigned. The ISC is not available when the operating system attempts to open an ISC file. The ISC is currently in an assigned state. Another program is using the ISC.

8510

device_ state_ unit_ reserved. The ISC is not available when the operating system attempts to open an ISC file. The ISC is currently in an reserved state.

8512

device_ state_ invalid. The ISC is not available when the operating system attempts to open an ISC file. The ISC is currently in an reserved state.

8514

memory_ not_ expanded. The operating system has obtained an external buffer while trying to open an ISC file, but is unable to obtain enough memory to expand the buffer.

8516

no_ path_ to_ hub. The operating system is unable to communicate with the hub as it attempts to open an ISC file. The operating system has tried all available paths to the hub. This TRAK code does not indicate the reason for the communication failure.

8518

isc_ open_ successful. The operating system has successfully opened an ISC file. The ISC IOAT entry is inserted in the TRAK buffer.

8520

isc_ ir_ start_ of_ close. The Default Read independent runner associated with this ISC file is starting to close the ISC file.

8522

isc_ ir_ close_ complete. The operating system has successfully closed an ISC file.

8524

isc_ ir_ error_ on_ read. The ISC Default Read independent runner is reporting an error condition during an attempt to read information from an ISC file.

8526

isc_ ir_ dfr_ fired. The ISC Default Read independent runner has initiated a default read operation.

8528

isc_ ir_ dfr_ complete. The ISC Default Read independent runner has received an I/O complete on a default read operation. The I/O queue element and 50 bytes of the external buffer are inserted into the TRAK buffer.

8530

dfr_ cancel_ initiated. The operating system has initiated a conditional cancel on an active default read operation.

8532

dfr_ cancel_ complete. The conditional cancel initiated by the operating system has completed (not necessarily successfully). The I/O queue element is inserted in the TRAK buffer.

8534

io_ fired. The operating system has initiated the I/O operation requested by the user program. This TRAK code does not include default reads.

8536

io_ complete. The operating system has completed the I/O operation requested by the user program. The I/O queue element and the first 50 bytes of the user buffer are inserted into the TRAK buffer.

8538

start_ of_ close. A user program has issued a close BCT or the user program has been terminated by an external process. The ISC's IOAT entry is inserted into the TRAK buffer.

8542

program_ read_ amr. A user program has issued a read AMR request.

8544

program_ write_ amr. A user program has issued a write AMR request.

8546

write_ amr_ complete. The write AMR operation requested by a user program is completed.

8548

satisfy_ complex_ wait. A user program is waiting on a complex wait condition that includes an ISC operation. The user program is notified that information is available from the ISC.

8550

program_ read_ data. A user program has issued a request to read data from an ISC file.

8552

program_ write_ data. A user program has issued a request to write data to an ISC file.

8556

cwt_ query. The operating system is asking if the ISC portion of a complex wait condition has been satisfied. The IOAT entry of the ISC is inserted in the TRAK buffer.

8558

task_ cannot_ be_ terminated. The operating system is attempting to terminate a task because the task has encountered a very serious ISC error. The operating system is unable to terminate the task.

8560

satisfy_ dcom_ wait. A user program has issued a DCOM wait BCT. The information requested in the BCT has become available with the completion of a default read operation.

MAILBOX TRAK Codes (8800 Series)

8801

MAIL SEND - BEGIN.

8802

MAIL SEND - END - ALL ACKS.

8803

MAIL SEND - END ALL NAKS.

8811

MAIL RECV - MESSAGE.

8812

MAIL RECV - RSVP.

STOQ TRAK Codes (8900 Series)

8901

stoque module queue header entry.

DCM TRAK Codes (9500 Series)

These TRAK codes apply only to timesharing tasks that issue DATACOM BCTs.

9511

Enter Datacom WRITE. A timesharing handler has issued a DATACOM Write request. This TRAK code marks the beginning of the DATACOM Write path.

9512

Datacom WRITE completed. The operating system has completed a DATACOM write request. Control is transferred by to the timesharing handler program.

9521

Enter Datacom READ. A timesharing handler has issued a DATACOM Read request. This TRAK code marks the beginning of the DATACOM Read path.

9522

Datacom Read Completed. The operating system has completed a DATACOM write request. Control is transferred by to the timesharing handler program. The requested data has been placed in the TTY buffer.

9531

Enter ACQBUF. A timesharing handler has issued an ACQBUF BCT. The operating system is about to look into the MCS handler buffer to see if it contains any data intended for the timesharing handler.

9532

TSHFEP goes dozing. The operating system, running for a timesharing handler, has searched the MCS handler buffer for data intended for the timesharing handler. No data has been found. The operating system is about to doze for a specified period of time.

9533

TSHFEP comes back from dozing. The operating system, running for a timesharing handler has finished dozing and is about to determine the next step in processing.

9534

Set address of the Handler Buffer to be serviced.

9535

Exit ACQBUF. The operating system has finished searching for data in MCS handler buffer (including doze periods). Control is returned to the timesharing handler.

9541

Enter COMBUF. A timesharing handler program has issued a COMBUF BCT to notify the operating system that information from a station has been placed in the MCS handler buffer.

9542

Exit COMBUF. The operating system has completed processing of the COMBUF BCT. If a station was waiting for data and data for the station was present in the MCS handler buffer, the data has been transferred into the TTY buffer of the program corresponding to the station.

DCU TRAK Codes (9600 Series)

9601

DCP - START OF SET DCP.

9602

DCP - UNSUCCESSFUL SET OPTION.

9610

DCP - MARK RJE STATION.

9620

DCP - START OF RESET OPTION.

9621

DCP - RETURN DCP TABLE MEM.

DCP TRAK CODES (9700 Series)

9700

DCP FILE OPEN - START OF DCP OPEN.

9702

DCP FILE OPEN - LOAD HOST REQUIRED.

9704

DCP FILE OPEN - END OF DCP OPEN.

9706

DCP FILE CLOSE - START OF CLOSE.

9708

MCS READ BCT.

9710

DCP LOGICAL READ I/O COMPLETE. — (HDR).

9712

MCS WRITE BCT..

9714

DCP WRITE BCT.I/O COMPLETE. — — — (HDR).

9716

START OF DCP MODULE LOAD..

9718

START OF DCP MODULE UNLOAD..

9720

MCP WRITE TO STATION. — — — — — — (HDR).

9722

DCP STATUS BEGIN.

9724

FOUND A DCP TO STATUS..

9726

B874 DEFAULT READ FIRED..

9728

WRITE RESULT HDR RETURNED (B874) — (HDR).

9730

B874 READ IO COMPLETE. — — — — — (HDR).

9732

B874 RJE INPUT RECEIVED. — — — — (HDR).

9734

B974 TEST I/O — — — — — — — — — — (HDR).

9736

STATUS CHANGE / MCP WRITE. — — — (HDR).

9738

B974 MCS WRITE I/O..

9740

B974 SET OPTION.WRITE LR. — — — — (HDR).

9742

B974 WRITE RR. — — — — — — — — — (HDR).

9744

B974 DEFAULT READ FIRED..

9746

B974 CANCEL OF DEFAULT READ IOC..

9748

B974 DEFAULT READ I/O COMPLETE..

9750

SENDING A NACK TO A B974. — — — (HDR).

9752

BAD B974 HEADER RECEIVED. — — — (HDR).

9754

B974 HEADER IGNORED. — — — — — — (HDR).

9756

B974 RJE INPUT RECEIVED. — — — — (HDR).

9758

CANCEL DEFAULT READ..

9760

UNABLE TO CANCEL DEFAULT READ.

9762

START OF CHANGE MCS ID..

9764

FIRE MCP DCP I/O. — — — — — — — (Q1).

9766

DCP I/O COMPLETE. — — — — — — — (Q2).

9768

B874 FLUSH QUEUES START..

9770

29 HEADER RETURNED TO MCS. — — (HDR).

9772

INVALID RESULT HDR RECEIVED — — (HDR).

9774

MCS STATUS CHANGED.

IOM TRAK Codes (9900 Series)

9900

I/O QUEUED.

9901

I/O FIRED.

9902

CANCEL I/O FIRED.

9903

CNCL I/O NOT FIRED, NO IN_ PRC Q.

9910

I/O COMPLETE, NO EXCEPTION.

9911

I/O COMPLETE WITH EXCEPTIONS.

9912

I/O CNCLD BY UNCONDITIONAL CNCL.

9913

QUEUED I/O FIRED.

9914

RJE I/O COMPLETE - NON RJE HDW.

9920

SHARED LOCK.

9921

SHARED UNLOCK.

9922

PURGED LOCK.

SECTION 9
REMOTE SUPPORT

OVERVIEW OF REMOTE SUPPORT

The hardware and software of V Series systems can perform maintenance and diagnostic functions from an off-site (remote) location. These functions are available through the Maintenance Processor, a microprocessor-based subsystem that provides operator and maintenance interfaces to the off-site system's main processor. Remote access to the Maintenance Processor is allowed through a single data communications link that can only be activated from the local site.

NOTE
This section describes the remote support capabilities of a Unisys V 300 system. This material only applies to V 300 systems.

REMOTE CAPABILITIES

Remote capabilities function in two modes: Passive Mode and Active Mode. In Passive Mode, the functional (local) system controls the Maintenance Processor. In Active Mode, the Maintenance Processor controls the functional (local) system.

Remote capabilities interface to a TD830 or MT983 functioning in a point-to-point (contention) datacomm environment, although you can use any remote device capable of emulating these terminals and protocol. The remote device connects to the system through the Remote Link of the Maintenance Processor.

Passive Mode

In Passive Mode, the Maintenance Processor operates in On-Line Mode, and any of its I/O ports can be declared as an I/O Resource of the functional system.

In this role, the functional system directs the Maintenance Processor, and the Maintenance Processor Remote Link Port is defined as an I/O Resource of the functional system. The remote device can communicate with either the MCP or a normal state program. The functional system must be operational for the remote passive capabilities to be utilized.

Active Mode

In Active Mode, the Maintenance Processor operates in Console Mode, and a remote device performs Maintenance Processor Command and Display functions. These functions resemble those of the local ODT. In this mode, only the Maintenance Processor, flexible disk drive, and a local ODT need to be operational.

The ODT at the customer's facility provides Command and Display functions for the Maintenance Processor when the Mode Panel switch is in the Console position. It provides these functions for the MCP when the switch is in the On-Line position.

MAINTENANCE PROCESSOR / SYSTEM RELATIONSHIP

Figure 9-1 shows the architectural relationship between the system and the Maintenance Processor for a V 300 system. Programs loaded from the flexible disk drive determine which functions the Maintenance Processor can perform and which I/O ports are used.

All remote capabilities use the Remote Port of the Maintenance Processor, as shown connected to the Modem/Acoustic Coupler in figure 9-1.

Remote support involves three other Maintenance Processor I/O ports :

- the Host Console Port
- the Test Bus Interface Port
- the Data Link Processor Port

Host Console Port (HCP)

The HCP is the Maintenance Processor interface to the System Maintenance Controller. This port allows the MCP to perform all active maintenance functions to the functional system. Such functions include controlling system clock distribution, setting and resetting state devices, and driving and monitoring system diagnostic tests.

The HCP interface is primarily driven by the Maintenance Processor when the functional system is idle. However, some functions, such as Maintenance Panel setup, can be performed while the functional system is running. When the system is running, the Maintenance Processor continually monitors the port for a change in system status.

Test Bus Interface (TBI) Port

The TBI port is the Maintenance Processor interface to the UIO DLP Base Maintenance Card. This port allows the Maintenance Processor to perform all active maintenance functions on any selected DLP.

The Maintenance Processor drives this port regardless of the state of the functional processor; however, the DLP under diagnostic test must not be available to the system as an I/O Resource.

Data Link Processor (DLP) Port

The DLP port is the path by which the functional system uses the Maintenance Processor and any of its I/O ports as an I/O Resource.

Two requirements must be met in order to use a Maintenance Processor port as a system I/O Resource:

- the Maintenance Processor I/O port must be declared to the MCP by a UNIT command
- the Maintenance Processor must contain a flexible disk program that will provide a logical path from the UC-DLP to the Maintenance Processor port.

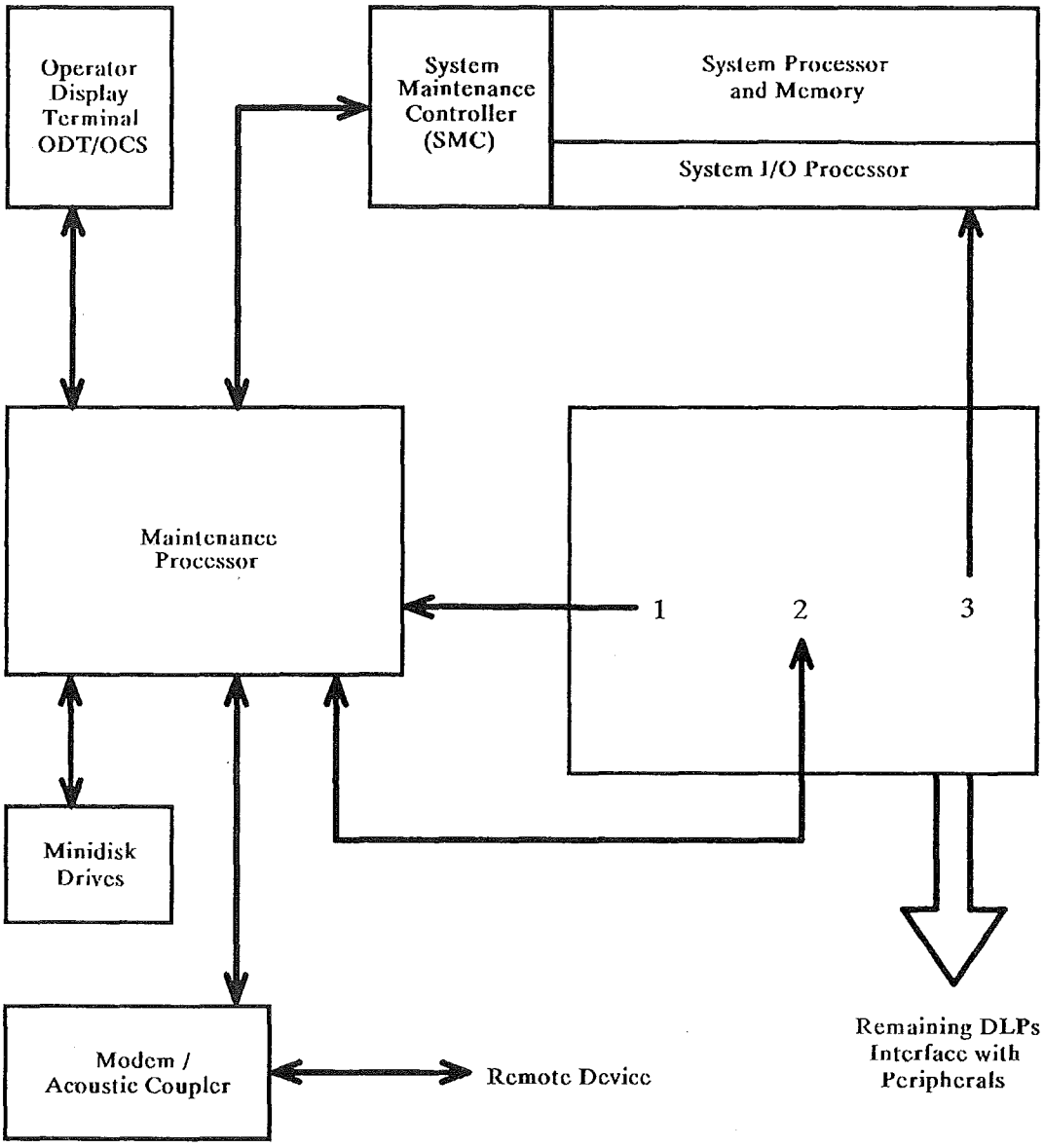


Figure 9-1. System / Maintenance Processor Relationship

The numbers in Figure 9-1 represent the following information:

1 DLP Base Distribution Card
2 UC - DLP
3 DLP Base Maintenance Card

CUSTOMER SITE REQUIREMENTS FOR REMOTE SUPPORT

Passive Mode

Special software enables the specialist at a remote site to analyze files interactively through the Maintenance Processor or universal console at the customer site over a data communications link. Communication with the system to be analyzed is done from a terminal on the specialist's own system, which is assigned to this software in a GEMCOS network. Table 9-1 summarizes these software programs and their functions.

The files, contained on the release tape, must be installed on the system to be analyzed. The programs on this tape are run at the local site and accessed by the specialist at the remote site over the communications link. Once the remote link is active, the remote system can communicate with the local programs as if it were directly dialed into the local system. Figure 9-2 shows the hardware required to establish this link.

Table 9-1. Remote Functions Provided by the Passive Remote Software

Module	Remote Functions
RDGDOC	Release letter in printer backup (disk)format
RDGMAN	Section 10 of the V300 Maintenance User's Guide in printer backup (disk) format
MPINT	Provides an interface between the Maintenance Processor or the universal console of a V Series system and any data communicationsprogram capable of communicating through CRCR with a GEMCOS-generated MCS Provides all of the basic functions of anMCS
MPCNDE	Provides an interface between the Maintenance Processor or the universal console of a B 2900/B 3900/B 4900 system and the CANDE TSHFEP program Used instead of the MPINT program if the specialist at the remote site wishes to access any CANDE software at the local site.
RDCODE	Copies a file containing hexadecimal codeson disk or pack to a new file and either Encodes the file before it is transmitted or Decodes the file after transmission
RDCOPY	Transfers files between the two systems
RDMLOG	Allows the specialist to access the Maintenance Log interactively

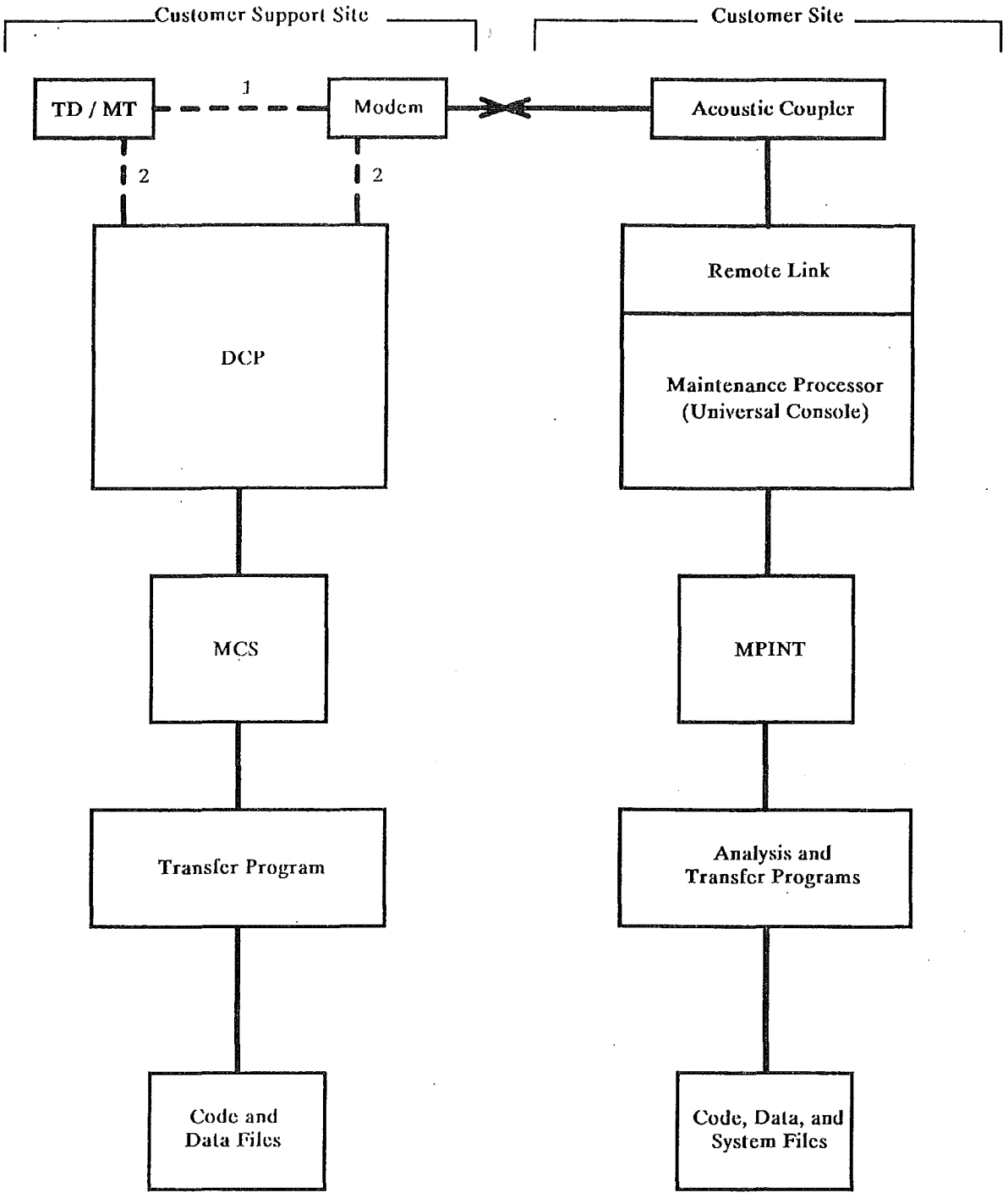


Figure 9-2. Site Requirements for Establishing a Remote Passive Link

The numbers in figure 9-2 represent the following information:

1 Standard Implementation
2 Enhanced Implementation

Active Mode

PANRMV, PANSMV, MODTST, and TSTGEN flexible disks contain remote active capabilities. Any active functions performed locally can be performed from the remote device if the proper flexible disk is loaded and the remote link is active. Table 9-2 summarizes the functions provided with each flexible disk.

Table 9-2. Remote Functions Provided by the Active Remote Flexible Disks

Module	Remote Functions
PANRMV and PANSMV	Interrogation and modification of all and selected, visible, and hidden states, and system parameters
	Control of the system clocks
	Control and display functions in the initiation and monitoring of released stand-alone maintenance tests
	Control and display of the Maintenance Panel functions
	Writing and reading of memory with MCP base relative or absolute memory addresses
	Writing and reading memory with virtual addresses (PANSMV only)
	Singe Instruct and Single Clock operation
	Loading and verification of Control Store (PANRMV only)
	Halt-Loading
MODTST	Initiation and monitoring of Chain tests, RAM tests, memory tests, and path tests
TSTGEN	Recording of the system operation including setup, initiation, and monitoring

Flexible disks with remote capabilities also let the local ODT communicate with the remote device without the Maintenance Processor performing any undesired function. Someone at the remote device can interrogate, instruct, and receive replies from the person at the customer site without voice communication. No other special software is required at the customer site.

Figure 9-3 shows the hardware requirements for establishing the remote active link.

The modem or acoustical coupler used will vary, because of differences in telephone services worldwide. The Maintenance Processor's remote port provides an RS232/CCINT interface capable of functioning in synchronous or asynchronous mode at rates of 1200, 2400, or 4800 baud.

Figure 9-4 shows the Maintenence Processor's DRI card connection to the modem or acoustic coupler. The system is shipped with the DRI card's J2 port connected by a ribbon cable to the remote paddle board.

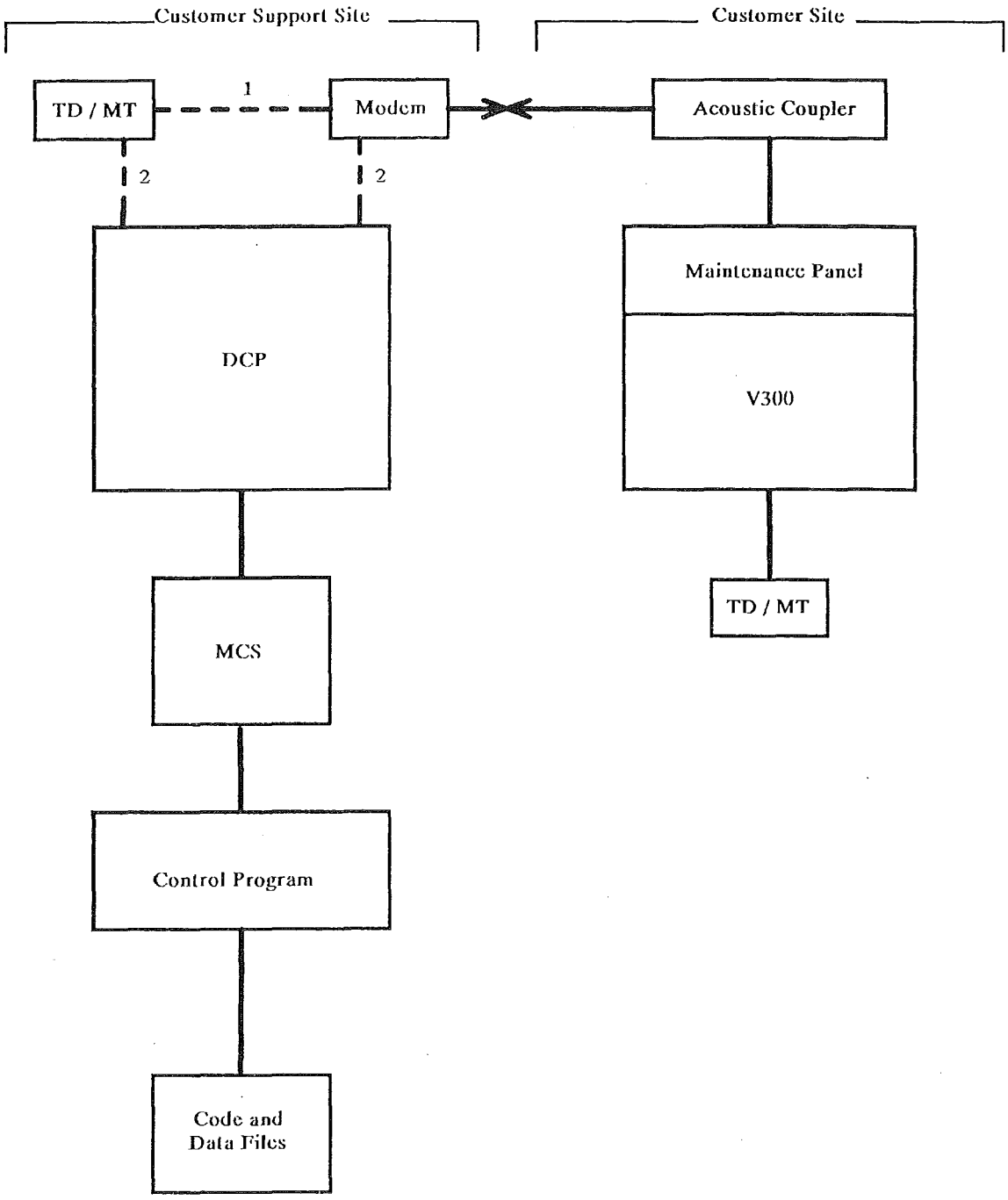


Figure 9-3. Site Requirements for Establishing a Remote Active Link

The numbers in figure 9-3 represent the following information:

1 Standard Implementation
2 Enhanced Implementation

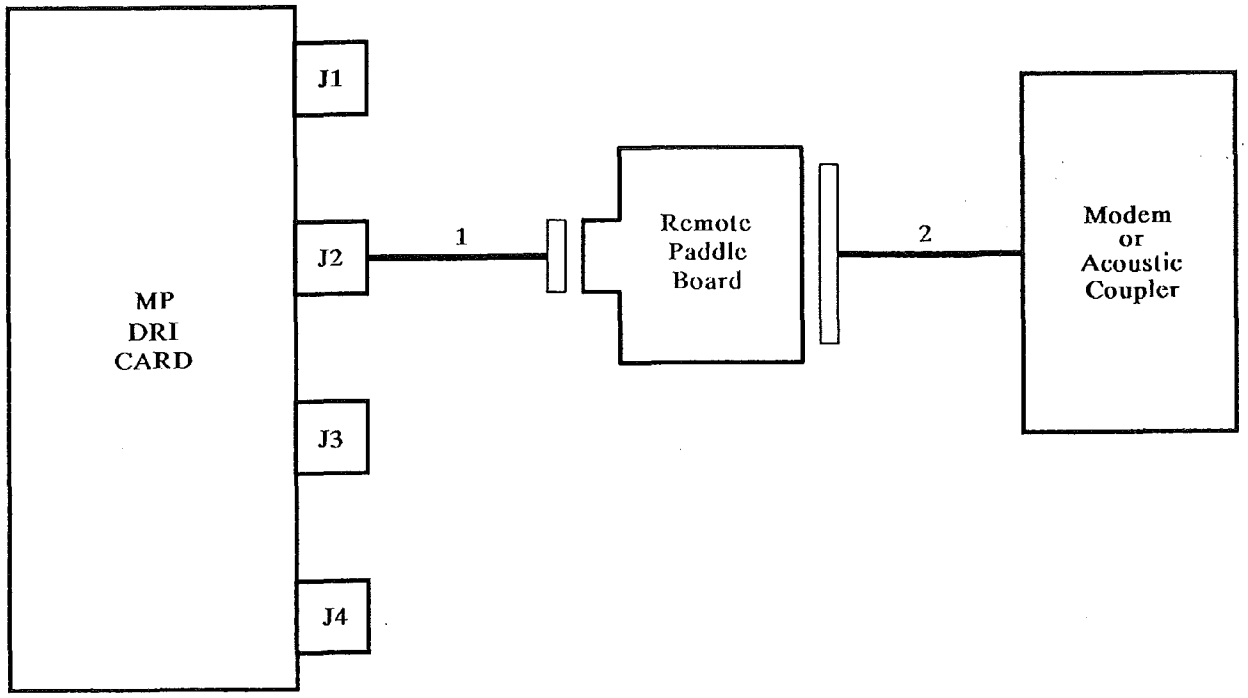


Figure 9-4. The DRI Card - Modem/Acoustic Coupler Connection

The numbers in figure 9-4 represent the following information:

1 Ribbon Cable
2 50 W/E Data Set Cable

APPENDIX A
OTHER SYSTEM INFORMATION

This appendix contains smaller portions of reference material relating to the MCP/VS operating system. It also contains material dealing with MCP/VS source code listings.

Much of the reference material deals with V Series hardware, specifically with V 300 systems. The hardware-related material in this appendix applies only to V 300 systems. The authoritative sources for V 300 hardware information are the *V 300 System Reference Manual* and the *V 300 Maintenance Panel Users Guide*. Most of this information appears in one of these documents (occasionally in a different form). Use the *V 300 System Reference Manual* and the *V 300 Maintenance Panel Users Guide* for definitive hardware information.

The following material is included in this appendix:

- MCP/VS Reference Material
  - Op Code List
  - Predicted Branch Table
  - Lock/Event Format
  - System Status Format
  - Instruction Variant (AF/BF) Format
  - Address Controller / Index Register / Fetch Scratch Pad (FSP) Relationships
  - Address Error Table
  - Invalid Instruction Table
    Invalid Field Comparison
    Invalid Operand Field
  - Processor Result Descriptor Table
  - Fetch/XM Error Handling Interface
  - Reader Scratch Pad (RSP) Assignments
  - Sub-MCP Base (Absolute Memory) Assignments
  - V 300 IOP Scratch Pad Assignments
  - V 300 Supported DLPs
  - BCT Summary
  - Software—Generated Result Descriptors (I/O Module)
  - Pre-term Code List
- MCP/VS Support Tools
  - LSTMOD (MCP/VS Source Listing Utility)

MVP/VS OP CODES

Table A-1 lists the instruction set for V 300 systems.

Table A-1. V 300 Instruction Set Summary

MNE	OP Code	AF Addr	BF Addr	# of	Remarks
INC	01	aa	bb	2	B < — A + B
ADD	02	aa	bb	3	C < — A + B
DEC	03	aa	bb	2	B < — B - A
SUB	04	aa	bb	3	C < — B - A
MPY	05	aa	bb	3	C < — A * B
DIV	06	aa	bb	3	C < — B/A (Remainder in B)
	07				<Unused>
MVD	08	vv	--	3	B < — A (size = C - B)
MVL	09	nn	--	3	rotate A, B & C
MVA	10	aa	bb	2	B < — A (left just, space fill)
MVN	11	aa	nn	2	B < — A (right just, zero fill)
MVW	12	nn	nn	2	B < — A
MVC	13	nn	nn	2	B < — A; Clear A
MVR	14	aa	nn	2	B < — A repeated rr times
TRN	15	nn	nn	3	C < — B indexed by A
SDE	16	aa	bb	2	38 < — loc of hit
SDU	17	aa	bb	2	38 < — loc of miss
SZE	18	aa	bb	2	38 < — loc of zone hit
SZU	19	aa	bb	2	38 < — loc of zone mis;
NOP	20		1		No operation
LSS	21		1		Branch if LOW
EQL	22		1		Branch if EQL
LEQ	23		1		Branch if LOW or EQL
GTR	24		1		Branch if HIGH
NEQ	25		1		Branch if not EQL
GEQ	26		1		Branch if EQL or HIGH
BUN	27		1		Branch always
OFL	28		1		Branch if OVF = 1
HBR	29		1		Halt dep on addr 48, then branch
NUL	2A		1		Branch if NULL
GTN	2B		1		Branch if HIGH or NULL

Table A-1. V 300 Instruction Set Summary (Continued)

MNE	OP Code	AF Addr	BF Addr	# of	Remarks
BCT	30	AA	AA	0	Save state, fake HCL
NTR	31	nn	nn	1	Save SCW & params, branch
EXT	32			1	Restore SCW, branch back
BRT	33	aa	mm	1	reset mm in A
BST	34	aa	mm	1	Set mm in A
VEN	35	cc	cc	2	Save SCW & parms, switch PT and branch
	36				<Unused>
SLL	37	aa	nn	2	IX1 < — hit + D.O.
SLD	38	aa	nn	2	IX1 < — hit + D. IX2 < — Prev Link + D.O.
SEA	39	aa	nn	3	IX1 < — hit + D.O.
BZT	40	aa	mm	1	compare mm in A
BOT	41	aa	mm	1	compare ~mm in A
AND	42	aa	bb	3	C < — A and B
ORR	43	aa	bb	3	C < — A or B
NOT	44	aa	bb	3	C < — A excl or B
CPA	45	aa	bb	2	UA compare A & B
CPN	46	aa	bb	2	UN compare A & B
SMF	47	mi	ii	0	No operation
HBK	48	ii	mm	0	Halt based on mm, 46 & 48
EDT	49	aa	bb	3	C < — A formatted by B
IAD	50			1	Acc < — Acc + A
IAS	51			1	Acc, A < — Acc + A
ISU	52			1	Acc < — Acc - A
ISS	53			1	Acc, A < — Acc - A
IMU	54			1	Acc < — Acc * A
IMS	55			1	Acc, A < — Acc * A
	56				<Unused>
IMI	57			1	A < — A + 1
ILD	58			1	Acc < — A
IST	59			1	A < — Acc

Table A-1. V 300 Instruction Set Summary (Continued)

MNE	OP Code	AF Addr	BF Addr	# of	Remarks
LOK	60		vv	1	UNLK,LOKU or LOKC (00,01,02) SGNL,WAIT or TEST (04,05,06) SET (07)
ASP	61	ii	ii	1	TOPSTK < — TOPSTK + A
HCL	62	AA	AA	2	Save state + B data on MCP stack, Branch to A ENVP
RET	63				Restore state from VEN,HCL,HHC
SLT	64		vv	3	Search List
WHR	65		vv	1	Communicate To Hardware
STB	66		vv	3	Search Table
LIX	67		vv	1	Load Index Registers
SIX	68		vv	1	Store Index Registers
CRE	69	01	00	2	Initialize Event
CRL	69	04	01	2	Initialize Lock
MEVC	6A	06	00	2	Move Count Field from Event Structure
MVLR	6A	04	01	2	Move Lock Owner's Task Number
RAA	70			1	Acc < — Acc + A
RAS	71			1	Acc, A < — Acc + A
RSU	72			1	Acc < — Acc - A
RSS	73			1	Acc, A < — Acc - A
RMU	74			1	Acc < — Acc * A
RMS	75			1	Acc, A < — Acc * A
RDV	76			1	Acc < — Acc/A
RDS	77			1	Acc, A < — Acc/A
RLD	78			1	Acc < — A
RST	79			1	A < — Acc
ACM	84	00		0	Normalize Accumulator
	84	10		0	Convert real to integer
	84	20		0	Set mantissa sign to +
	84	30		0	Set mantissa sign to –
	84	40		0	Complement mantissa sign
	84	50		0	Clear Accumulator to -99 + 0
	84	6n		0	Increment exponent by n
	84	7n		0	Decrement exponent by n

Table A-1. V 300 Instruction Set Summary (Continued)

MNE	OP Code	AF Addr	BF Addr	# of	Remarks
CIO	85	aa	ii	2	Convert I/O Desc A into B
ATE	86	ii	vv	2	Alter Table Entry
MOP	87	aa	bb	2	MREG < — A masked by B
D2B	88	aa	bb	2	B(bin) < — A(dec)
B2D	89	aa	bb	2	B(dec) < — A(bin)
INT	90	aa	vv	1	A data to kernel, interrupt to routine vv
SRD	91	AA	AA	0	IX1 < — address R/D
RAD	92	im	cc	1	Read address from channel cc
BRV	93	aa	—	1	Reinstate Task pointed to by IX1
IIO	94	ii	cc	1	Initiate I/O Desc A
RDT	95	ii	ii	1	A < — timer
	96				<Unused>
STT	97	ii	ii	1	Set System clock/date
IOC	98	06	cc	2	I/O Complete
SST	99	—	vv	1	Obtain system status
MVS	A0	ss	dd	2	Move String from A to B
CPS	A1	ss	dd	2	Compare string A to B
HSH	A2	ss	dd	2	Hash String A to value B
FAIL	AB				System Failure
	Bx			1	Predicted Branch taken — > not taken
	Ex			1	Predicted Branch taken — > taken
	Fx			1	Predicted Branch taken — > taken

PREDICTED BRANCHES

Table A-2 shows the predicted branch values inserted in the most significant digit of a branch instruction op code. Examining predicted branches can be useful in determining code flow where there are no other obvious indications of the path taken through a section of code. This information is encountered in a memory dump.

The most significant digit of a branch op code is modified each time the branch is executed. The value of the most significant digit shows whether the branch was taken the last time it was executed. It also shows whether the hardware predicted that the branch would be taken (based on the previous execution of the branch).

Table A-2. Predicted Branches

OP MSD	Last Branch	Predicted
2x	NOT TAKEN	NOT TAKEN
Bx	TAKEN	NOT TAKEN
Ex	NOT TAKEN	TAKEN
Fx	TAKEN	TAKEN

LOCK/EVENT FORMAT

Figure A-1 shows the format of a lock structure in memory and the format of an event structure in memory.

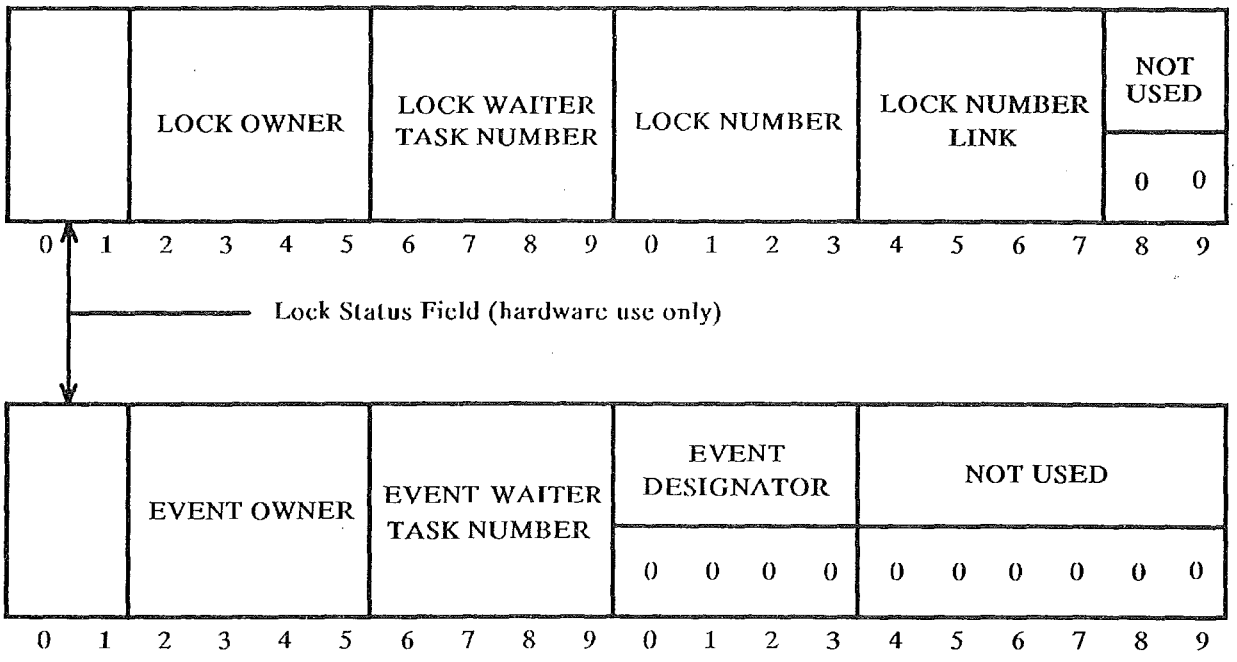


Figure A-1. Lock / Event Structure in Memory

SYSTEM STATUS

Figures A-2 and A-3 show the format of responses to the Obtain System Status instruction (OP 99). The format changes depending on the contents of the BF address of the instruction.

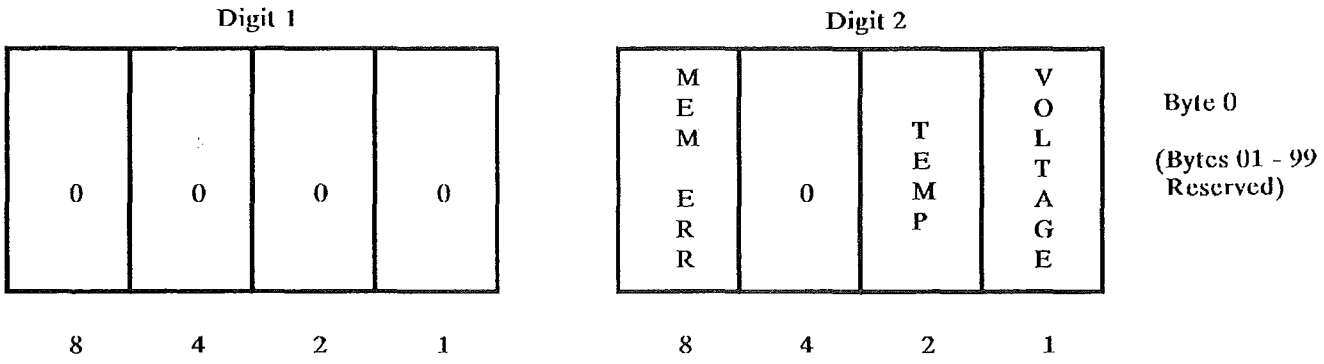


Figure A-2. System Status Response (BF = 00)

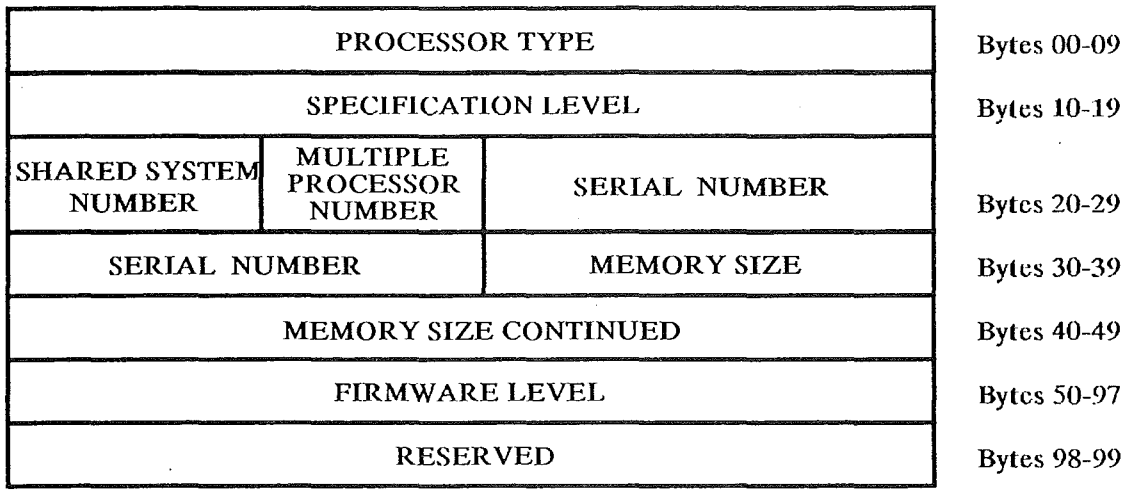


Figure A-3. System Status Response (BF = 01)

INSTRUCTION VARIANT (AF/BF) FORMATS

Table A-3 shows the instruction variant format used when the value in the A address field is data, rather than an address.

Table A-3. AF Format When A Address Is Literal

	Most Sig. Digit	Least Sig. Digit
Bit 8	1	S
Bit 4	0	L
Bit 2	1	L
Bit 1	A	L

In this table:

- 1, 0, and 1 in the most significant digit indicate that the value is data, not an address.
- A and S indicate the type of the data:

A	S	Data Type	Size
0	0	UN	(6 digits max.)
0	1	SN	(1 sign digit, 5 data digits max.)
1	0	UA	(3 characters max.)

- L, L, and L indicate the length of the literal.

Table A-4 shows the instruction variant format used when the A address field or the B address field is an indirect field length.

Table A-4. AF/BF Format When A/B Address Is Indirect Field Length

	Most Sig. Digit	Least Sig. Digit
Bit 8	1	U
Bit 4	1	U
Bit 2	T	U
Bit 1	T	0

In this table:

- 1 and 1 in the most significant digit indicate that an indirect field length is specified. The final field length is located at the base-relative address indicated by T, T, U, U, and U.
- T and T indicate the tens value of the base-relative address where the final field length is located.
- U, U, and U indicate the units value of the base-relative address where the final field length is located.
- Bit 1 of the least significant digit must be zero.

ADDRESS CONTROLLER / INDEX REGISTER / FETCH SCRATCH PAD

Table A-5 shows the relationship between the address controller digit of an instruction address, the corresponding index register(addressing mode) and the Fetch Scratch Pad (FSP) word where the index register is located.

Table A-5. Address Controller / Index Register / Fetch Scratch Pad

Index Digit	Extension Digit	Addressing Mode	FSP
0	None	Context Relative (0 = data, 1 = code)	—
4	None	IX1	9
8	None	IX2	A
C	None	IX3	B
0	C	Context Relative (0 = data, 1 = code)	—
4	C	IX1	9
8	C	IX2	A
C	C	IX3	B
0	D	Code Base Relative (data ref to code page)	—
4	D	IX4	C
8	D	IX5	D
C	D	IX6	E
0	E	<invalid>	—
4	E	IX7	F
8	E	<invalid>	—
C	E	<invalid>	—
0	F	<invalid>	—
4	F	<invalid>	—
8	F	<invalid>	—
C	F	<invalid>	—

PROCESSOR RESULT DESCRIPTORS

Table A-6 lists the processor result descriptors for V 300 Series processors.

Table A-6. Processor Result Descriptors

Processor R/D	Description
2000000000	Invalid Arithmetic
08000000**xx**	Invalid Instruction
020000**xx**00	Address Error
0100000000	Instruction Timeout
8000000000	Hard Memory Area Fault
0004000000	Accumulator Trap
0080000000	Stack Overflow
1000000000	Soft Memory Area Fault
4000000000	Trace Fault
0010000000	Soft Fault
0400000000	Memory Parity Error (Multiple Bit)

ADDRESS ERROR CODES

Table A-7 lists the possible values returned in the processor result descriptor for an address error.

Table A-7. Address Errors

Extended-Digits	Description
00	Address Error nonspecific
01	Invalid Address relationship (MVS-strg begin addr > strg end addr) (CIO-area begin >= area end)
02	HCL Function Limit error
03	Odd operand address
04	Invalid MAT entry most signif. digit
10	Index Register Error nonspecific
11	Invalid IX-Arithmetic (sum >= 10,000,000) (difference < 0)
12	Undigit in IX Register
13	Undigit in IX Indicant
14	IX3 negative on RET instruction
15	IX3 odd on RET instruction
20	Base/Limit Error nonspecific
21	Limit error in instruction Fetch
22	Limit error in Addr Resolution (IA)
23	Write Limit error from operand processing
24	Read Limit error fromoperand processing
30	Undigit in Address nonspecific
31	Undigit in Instruction Fetch address
32	Undigit in Address Resolution
33	Undigit in Operand Write address
34	Undigit in Operand Read address
40	Branch address error nonspecific
41	Branch address > limit
42	Undigit in Branch address
43	Odd Branch address

Table A-7. Address Errors (Continued)

Extended-Digits	Description
50	Invalid EN or MAN in operands nonspecific
51	Invalid EN - nonspecific
52	MSD EN = A,B,C,E,F
53	Undigit in index portion of EN
54	Invalid MAN (undigit)
56	EN or MAN out of range
57	EN out of range
58	MAN out of range
60	Invalid EN or MAN IN MAT - nonspecific
61	Invalid EN - nonspecific
62	MSD EN = A,B,C,E,F
63	Undigit in index portion of EN
64	Invalid MAN (undigit)
66	EN or MAN out of range
67	EN out of range
68	MAN out of range

INVALID INSTRUCTION ERROR CODES

Table A-8 lists the possible values returned in the processor result descriptor for an invalid instruction error condition.

Table A-8. Invalid Instruction Errors

Extended Digits	Description
00	Invalid Instruction - nonspecific
01	Invalid Op code
02	Priv mode violation
03	Invalid address controller
04	Stack Overflow (NTR)
05	Counter overflow/underflow (CIO-MAST I/Os in progress > 4 dig) (IOC-MAST I/Os in progress < 0)
06	Invalid field comparison (See next page for conditions causing this)
07	Invalid operand field (See next page for conditions causing this)
20	Invalid AF or BF - nonspecific
21	Literal not allowed
22	Invalid Literal combination
23	Invalid INFL
24	Invalid variant (AF or BF)
25	Invalid AF variant
26	Invalid BF variant
31	Unauthorized access to Priv code (VEN/RET [MSD EN=D] * PRIV/)
32	Unauthorized access to Priv data (STRINGS [EN=0 or MSD EN=D]*PRIV/)
35	Attempt to modify ORIGINAL or FAULT entry (ATE)
36	Copy protection violation (ATE)
37	Protection Digit violation (HCL,HHC protection digits NEQ 'DD') (RET invalid stack frame indicator)

Invalid Field Comparison

When an invalid instruction error code is returned in the processor result descriptor, one of the possible errors is called invalid field comparison. Table A-9 lists some of the conditions that can cause this error to occur.

Table A-9. Invalid Field Comparison Errors

Extended Digits	Description
06	Invalid Field Comparison
	(ILS BF=2 event designator field NEQ 0)
	(LOCK-UNLOCK lock owner field NEQ task#) (LOCK-UNLOCK lock struc lock# NEQ MCP canonical #) (LOCK-UNCONDITIONAL lock struc lock# < MCP canonical #) (LOCK-CONDITIONAL lock struc lock# < MCP canonical #)
	(LOCK-EVENT CAUSE event designator field NEQ 0) (LOCK-EVENT WAIT event designator field NEQ 0) (LOCK-EVENT SET event designator field NEQ 0) (LOCK-CAUSE & RESET event designator field NEQ 0) (LOCK-RESET & WAIT event designator field NEQ 0)
	(MLS BF=0 event designator field NEQ 0) (MLS BF=1 LOCK-CANONICAL number = 0000)

Invalid Operand Field

When an invalid instruction error code is returned in the processor result descriptor, one of the possible errors is called invalid operand field. Table A-10 lists some of the conditions that can cause this error to occur.

Table A-10. Invalid Operand Field Errors

Extended Digits	Description
07	Invalid Operand Field
	(ATE - task number too large in ATE variant 2)
	(CIO - invalid descriptor OP code) (CIO - undigit in B/L pair, MAST # orMAST # out of range)
	(EDT - invalid operator) (HBR/HBK - invalid halt digit at address 48) (IIO - invalid descriptor OP code)
	(ILS BF=1 - undigit in Canonical number or Canonical number = 0000)
	(ILS BF=2 - undigit in Waiter field) (LOCK - undigit in any lock structure fields) (MLS BF=0 - EVENT-COUNT contains undigit) (MLS BF=1 - undigit in any lock structure fields)
	(MVS - move of odd length) (SLT - invalid descriptor fields other than address field) (SST - invalid shared system # provided by hardware) (STB - invalid descriptor fields other thanaddress field) (STB - invalid address relationship in C fielddescriptor) (STT - undigit in Time of Day) (WHR - undigits in address or address mod error)

FETCH/XM ERROR HANDLING INTERFACE

Table A-11 shows the error handling interface between the Fetch Module and the Execute Module (XM) of the V 300 hardware. When the Fetch Module detects an error, an error code is passed to the XM in a particular portion of sub-MCP memory called the fetch page. The 10 digits of the fetch page are numbered as follows:

D9 D8 D7 D6 D5 D4 D3 D2 D1 D0

Two types of error codes are passed, depending on whether or not the instruction causing the error was a branch or non-branch instruction.

Table A-11. Fetch/XM Error Handling Interface

A) Branch OPs including OP 20-29, OP 31-32, and OP 35-63. The Error Flag Digit is D7 in the Fetch [ASYL]. (To see this information, enter EFPAGE 0 0 8 through the maintenance processor.)

"0" = No Errors
"1" = Undigits in Memory Read Address
"2" = Invalid IX-Arithmetic
Sum >= 10,000,000 or Difference < 0
"3" = Limit Error Detected By Formatter
"5" = Undigits in IX Registers
"6" = Undigits in IX-Offset Syllable
"E" = Instruction Timeout
"F" = Unspecified Address Error

B) Non-Branch OPs or Fatal Errors in Branch OPs. Error Code Contained in D9-D6 in Fetch [OPSYL]. Digits D5-D0 contain original OPSYL. (To see this information, enter EFPAGE 1 0 8 through the maintenance processor.)

"1E10" = Invalid Instruction
"1E12" = Invalid AF or BF
"1E20" = Unspecified Address Error
"1E21" = Undigits in Formatter Read Address
"1E22" = Invalid IX-Arithmetic
Sum >= 10,000,000 or Difference < 0
"1E23" = Limit Error Caused By Formatter
"1E25" = Undigits in IX Register
"1E26" = Undigits in IX-Offset Syllable
"1E27" = Limit Error Caused By Reader
"1E28" = Undigits in Reader Read-Address
"1E41" = Instruction Timeout
"1E82" = Multi-Bit Parity Error

READER SCRATCH PAD (RSP) ASSIGNMENTS

Table A-12 shows the general uses of each portion of the reader scratch pad (RSP). To see this information, enter RSP 0 15 through the maintenance processor.

Table A-12. Reader Scratch Pad (RSP) Assignments

Base/Limit Entry	Comment
0-7	Variable Base/Limits (Hardware Base/Limits)
7	MCP Base/Limit (Hardware Base/Limit) Used for SRD only.
8	Temporary, Initialized to be 0000000000
9	Temporary, Initialized to be 99000LLLLL
A	MCP Data Area Base/Limit (Hardware Base/Limit)
B	MCP Environment Table (Hardware Address)
C	User Environment Table (Hardware Address)
D	Address of the User Services Memory Area Table (USMAT) and the # of Entries in USMAT (NNAAAAAAAA) (Hardware Address)
E	Active Environment Number, Interrupt Flags (IIIIIIHHMM)
F	Measurement (MOP) Register (DDDDDDNN00)

SUB-MCP BASE (ABSOLUTE MEMORY)

Table A-13 shows the layout of the first 10,000 digits of memory on V 300 systems. This memory is reserved for the use of the hardware and cannot be directly accessed by the operating system. To see this information use the MEM ABS command of the maintenance processor.

Table A-13. Sub-MCP Base (Absolute Memory)

Digits	Description
220–229	Memory Area Status Table Base Address
230–239	Unused
240–249	RLIST Base Address (Hardware Address)
250–269	Unused
270–279	# of Entries in MCP Environment Table (6 Digits, Right Justified)
280–289	# of Entries in User Environment Table (6 Digits, Right Justified)
290–299	Reinstate List Current Entry Pointer (Hardware Address)
300–307	Mobile IX4
308–315	Mobile IX5
316–323	Mobile IX6
324–331	Mobile IX7
332–339	Unused
340–345	Time of Day Timer (Part I — YYYYMMDD00)
350–359	Time of Day Timer (Part II — SSSSSSSS00)
360–369	Task Timer for Current Task (TTTTTT0000)
370–370	Memory Error Report Pending (Written)
371–379	Unused
380–389	SNAP Picture Report Address (Machine Base)
390–399	Memory Error Report Address
400–655	IOP Scratchpad Area, (See Separate Table)
656–688	IOP Mailbox (Part of IOP Scratchpad)
689–3999	IOP Scratchpad Area, (See Separate Table)
4000–9999	Unused

V 300 IOP SCRATCH PAD ASSIGNMENTS

Figure A-4 illustrates the layout of the IOP mailbox within the IOP scratch pad.

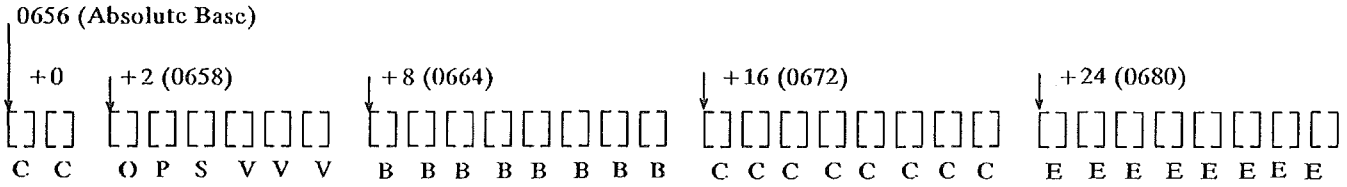


Figure A-4. IOP Mailbox

Figure A-5 shows the format of one word in the IOP scratch pad.

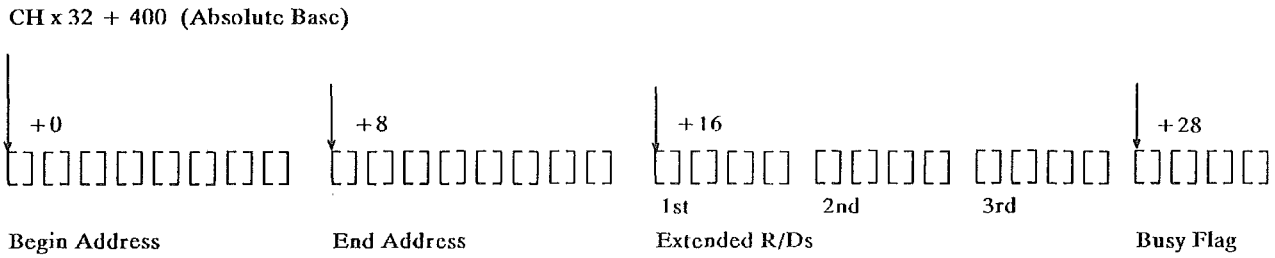


Figure A-5. IOP Scratch Pad Word Format

Table A-14 lists the absolute address in in sub-MCP memory where each channel's scratch pad begins.

Table A-14. Channel Scratch Pad Locations

Ch. No.	Begin Address	Ch. No.	Begin Address	Ch. No.	Begin Address	Ch. No.	Begin Address
00	0400	20	1040	40	1680	60	2320
01	0432	21	1072	41	1712	61	2352
02	0464	22	1104	42	1744	62	2384
03	0496	23	1136	43	1776	63	2416
04	0528	24	1168	44	1808	64	2448
05	0560	25	1200	45	1840	65	2480
06	0592	26	1232	46	1872	66	2512
07	0624	27	1264	47	1904	67	2544
10	0720	30	1360	50	2000	70	2640
11	0752	31	1392	51	2032	71	2672
12	0784	32	1424	52	2064	72	2704
13	0816	33	1456	53	2096	73	2736
14	0848	34	1488	54	2128	74	2768
15	0880	35	1520	55	2160	75	2800
16	0912	36	1552	56	2192	76	2832
17	0944	37	1584	57	2224	77	2864

V 300 SUPPORTED DLPS

Table A-15 lists the supported Data Link Processors (DLPs) for V 300 systems.

Table A-15. V 300 DLPs

No.	DLP Type	No.	DLP Type
00		30	Image Page Printer (IPP)
01	Card Reader	31	Seq. Host Transfer (Disk)
02	Train Printer	32	PE Streamer Tape
03		33	Buffered PRN (BPST-DLP)
04	PE Magnetic Tape	34	SMD (180-byte)
05	Card Punch	35	Ethernet (ICP-1)
06		36	
07	Universal Console	37	Memorex 3680 Disk
08	Host Transfer (STD)	38	
09	Floppy Diskette (ICMD)	39	GCR Streamer Tape
0A	5N Disk File	3A	SMD (100-byte)
0B	Reader/Sorter (4A)	3B	
0C	ODT	3C	
0D	GCR Magnetic Tape (TCU)	3D	SCSI
0E	9-Track NRZ	3E	
0F	Host Transfer (DCP)	3F	
10	7-Track NRZ	40	
11		41	
12	Host Transfer (NSTD)	42	
13	Buffered Printer	43	
14		44	
15		45	
16	HC-2 (ISC)	46	
17	B924 Buffered Printer	47	Uniline
18	UIO Line Expansion	48	SSP
19	UIO Line Expansion	49	
1A	UIO Line Expansion	4A	
1B	UIO Line Expansion	4B	
1C		4C	
1D	GCR Mag Tape (Formatter)	4D	ORS DLP
1E	FIPS Tape	4E	
1F	FIPS DISK	4F	

Table A-15. V 300 DLPs (Continued)

No.	DLP Type	No.	DLP Type
20		50	Telcom DLP
21		51	
22		52	
23		53	
24		54	
25		55	
26		56	
27	Laser Beam Printer	57	
28		58	
29		59	
2A		5A	
2B		5B	
2C		5C	
2D		5D	
2E		5E	
2F		5F	
		FE	QWIK Disk

BCT SUMMARY

The Branch Communicate (BCT) instructions of the MCP/VS operating system are summarized on the following pages. For more detailed information about a given BCT, see the *V Series Program Interfaces Programming Reference Manual.* Table A-16 summarizes the BCTs in numeric order.

Table A-16. Branch Communicates (BCTs) in Numeric Order

BCT	Function		Comment
BCT 0114	READ, LOCK		
	:8-:4 READ IGNORE LOCK	(00XX-0)	
	READ WITH LOCK,		
	no data transfer	(01XX-4)	
	READ WITH LOCK	(10XX-8)	
	READ,WAIT IF LOCKED	(11XX-C)	
BCT 0134	OPEN		
	OPEN IN	(0)	
	OPEN OUT	(1)	
	OPEN I/0	(2)	
	OPEN O/I	(3)	
	OPEN EXTEND	(4)	
BCT 0154	CLOSE		
BCT 0174	OVERLAY		
BCT 0194	STOP		
BCT 0214	TODAYS-DATE	(00)	MMDDYY
		(01)	YYMMDD
	TIME	(10)	
	MEMORY-SIZE	(20)	
	GROW	(21)	
	INTERROGATE DISK	(3X)	
	DATE	(40)	
	ARM	(5X)	
	MIXTBL	(6X)	
	TODAYS-NAME	(7X)	
	TIME-60	(8X)	
	INTERROGATE PACK	(9X)	
	PROCESSOR ID	(A0)	
	NEW MIXTBL	(DX)	

Table A-16. Branch Communicates (BCTs) in Numeric Order (Continued)

BCT	Function	Comment
BCT 0234	WRITE, UNLOCK	
	:8-:4 WRITE	(00XX-0)
	WRITE W/O UNLOCK	(01XX-4)
	UNLOCK	(10XX-8)
BCT 0254	ACCEPT	(0)
	DISPLAY	(1)
	WAIT	(2)
	SORT	(5)
	SORTRETURN	(6)
	DISPLAY-LINES	(8)
BCT 0274	ZIP	
BCT 0294	EXITROUTINE	
BCT 0314	LOCK WITH SEEK, SEEK	
	:8-:4 SEEK IGNORE LOCK	(00XX-0)
	LOCK WITH SEEK	(01XX-4)
	SEEK WITH LOCK	(10XX-8)
	SEEK,WAIT IF LOCKED	(11XX-C)
BCT 0334	TRACE	
	DUMP	
BCT 0354	DATACOMM	
	READ	(00)
	FILL	(01)
	WRITE	(02)
	WRITE-READ	(03)
	WRITE-TRANS-READ	(04)
	WRITE-READ-TRANS	(05)
	ENABLE	(06)
	INTERROGATE	(07)
	DISABLE ON NO DATA	(08)
	WAIT	(10)
	DISABLE	(11)
	INTERROGATEEND-TEXT	(12)
	TRANSLATE TABLE FETCH	(13)
	READ EXT	(20)

Table A-16. Branch Communicates (BCTs) in Numeric Order (Continued)

BCT	Function		Comment
	FILL EXT	(21)	
	WRITE EXT	(22)	
	WRITE-READ EXT	(23)	
	WRITE-TRANS-READ EXT	(24)	
	WRITE-READ-TRANS EXT	(25)	
	ENABLE EXT	(26)	
BCT 0374	MICR3A+4A		
	POCKET SELECT	(29)	
	START FLOW	(42)	
	DEMAND FEED	(43)	
	POCKET LITE/ ADVANCE BATCH/ICM	(44)	
	MICROFILM SLEW	(45)	
	LOGICAL READ	(46)	
	STATUS	(47)	
	CHARACTERISTICS	(48)	
	SELECT	(60)	(3A)
	READFLOW	(62)	(3A)
	READ	(63)	(3A)
	POCKET LITE	(64)	(3A)
	ADVANCE BATCH CTR	(66)	(3A)
BCT 0394	SPACE POSITION		
BCT 0414	CRCR		
BCT 0434	DIRECT I/O		
	DO IO	(0)	
	OPEN	(1)	
	CLOSE	(2)	
	DO IO and Inhibit Channel	(3)	
BCT 0454	QUICK DATE	(40)	
	QUICK TIME	(10)	
	QUICK TODAYS DATE	(20)	
BCT 0474	SPO MESSAGE		

Table A-16. Branch Communicates (BCTs) in Numeric Order (Continued)

BCT	Function	Comment
BCT 0494	STOQUE	
	FILL INTO TOP	(2)
	FILL FROM TOP	(3)
	FILL FROM POLL	(4)
	FILL INTO BOTTOM	(8)
	FILL FROM BOTTOM	(9)
BCT 0534	USER VALIDATE	000000
	USER CHANGE	000100
	USER CODE	000200
	CHANGE SECURITY	CHGSEC
	LABEL EQUATE	EQUATE
	USERCODE VALIDATE	GETUSR
	PASSWORD CHANGE	SAC006
	USER INTERROGATE	USRFOO
	ZIP SPO	ZIPSPO
	JOB INFORMATION	JOBINF
BCT 0574	RELINQUISH PROCESSOR	
BCT 0594	DIRECT READ	
BCT 0614	READ PORT	
	WRITE PORT	
BCT 0634	SET ATTRIBUTE	
BCT 0654	GET ATTRIBUTE	
BCT 0674	OPEN AVAILABLE	
	OPEN PORT	
	OPEN WAIT	
BCT 0694	CLOSE PORT	
BCT 0794	GET PARAMETERS	
	INTERROGATE PARAMETERS	
	PROCESS CALL	
	PROGRAM CALL	
	PROGRAM CANCEL	

Table A-16. Branch Communicates (BCTs) in Numeric Order (Continued)

BCT	Function		Comment
BCT 0874	PROGRAM RETURN	(90)	
BCT 0894	DATABASE		
	OPEN	(01)	
	CLOSE	(02)	
	CREATE	(03)	
	STORE	(04)	
	READ DIRECT	(05)	
	READ INDIRECT	(06)	
	POSITION	(07)	
	EQUATE PATH	(08)	
	INSERT	(09)	
	REMOVE	(10)	
	BEGIN TRANSACTION	(11)	
	END TRANSACTION	(12)	
	FREE	(13)	
	RETURN	(99)	
BCT 0994	COMPLEX WAIT		

SOFTWARE-GENERATED RESULT DESCRIPTORS (I/O MODULE)

Table A-17 shows the software-generated result descriptors (R/Ds) that are created by the physical input/output module (IOM) of the operating system. These R/Ds are sometimes referred to as "logical R/Ds".

Table A-17. Software-generated R/Ds (IOM Module)

Error	Meaning
9000	I/O Canceled (Unconditional Cancel)
9010	No R/D on Channel (Undetermined Soft R/D)
9020	No R/D on Channel (Overtime I/O)
9021	No R/D on Channel (I/O complete on busy channel)
9022	I/O Not Canceled (I/O already complete)
9023	No R/D on Channel (No response from IOP)
9040	Invalid I/O descriptor (Busy on IIO instruction)
9050	Invalid I/O Descriptor (Address too big)
9051	Write in Restricted Disk (Write in MCP)

PRE-TERM CODES

The operating system generates one set of pre-term codes in response to processor errors and a second set in response to program errors that are detected by the operating system. Table A-18 lists the pre-term codes for processor errors. Table A-19 lists the pre-term codes for program errors detected by the operating system. In these tables, <F-N> represents a file name, <P-N> represents a program name, and <ADR> represents a program base relative address.

Table A-18. Pre-term Codes For Processor Errors

Error	Meaning
C100	Instruction Timeout
C200	Address Error
C400	Memory Parity Error - No Program Error
C800	Invalid Instruction
E800	Invalid Arithmetic Data

Table A-19. Pre-term Codes for Errors Detected by Operating System

Error	Meaning
9010	INVALID CLOSE <ADR> INVALID FIB ADDRESS
9020	INVALID CLOSE <F-N> <ADR> FILE NOT OPEN
9030	INVALID CLOSE <F-N> <ADR> INVALID USER DISK HEADER
9040	INVALID CLOSE <F-N> <ADR> SMASHED FIB
9050	INVALID CLOSE <F-N> <ADR> INVALID FILE ID
9060	INVALID CLOSE <F-N> <ADR> INVALID LABEL USE ADDRESS
9070	INVALID CLOSE <F-N> <ADR> FILE SIZE EXCEEDED
9080	INVALID CRCR <ADR> INVALID SIZE
9090	INVALID DCOM I/O <F-N> <ADR> FILE NOT OPEN
90A0	<P-N> INVALID READ WRITE FUNCTION
90B0	<P-N> INVALID SUBPORT STATE FOR READ
90C0	<P-N> INVALID SUBPORT STATE FOR WRITE
9100	INVALID DCOM I/O <F-N> <ADR> INVALID PINGPONG ADDRESS
9110	INVALID DCOM I/O <F-N> <ADR> INVALID PHONE NUMBER
9130	INVALID ARITHMETIC DATA <ADR>
9140	EOF NO LABEL <F-N> <ADR>
9150	I/O ERR <F-N> <ADR> INVALID RESULT DESCRIPTOR
9160	<P-N> I/O ERROR <F-N> <ADR> UNASSIGNED ERROR
9170	INVALID I/O DESCRIPTOR <F-N> <ADR>
9180	INVALID I/O LIMIT <F-N> <ADR>
9190	MEM PAR ADDRESS <ADR>
91A0	<P-N> DRIVER INITIALIZATION FAILED — NO FIA
91B0	<P-N> MICR OPEN <F-N> <ADR> FAILED, UNABLE TO CREATE DRIVER
91C0	<P-N> FORCED TERMINATION — DRIVER FAILED
91D0	<P-N> CLOSE <F-N> <ADR> INVALID IN FLOW MODE
91E0	<P-N> INVALID INTERRUPT REQUEST <ADR>
91F0	<P-N> TIMER CORRUPTED
9200	PAR NO LABEL <F-N> <ADR>
9210	PROG OVLY READ ERR
9220	<F-N> INVALID OPR MTP REWIND
9230	INVALID READ <F-N> <ADR> FILE RESTRICTED OR NOT OPEN
9240	INVALID OPEN <ADR> INVALID FIB ADDRESS
9250	INVALID OPEN <F-N> INVALID OPEN I/O
9260	INVALID SEQUENTIAL OPEN <F-N> <ADR> AREA NUMBER 1 NOT ON BASE PACK
9270	INVALID OPEN <F-N> <ADR> FILE NOT CLOSED
9280	INVALID OPEN <F-N> <ADR> INVALID BNA REQUEST
9290	INVALID OPEN EXTEND <F-N> <ADR> NO WRITE RING
92A0	INVALID OPEN <F-N> <ADR> SYSTEM ERROR
92B0	INVALID OPEN <F-N> <ADR> NO USER MEMORY

Table A-19. Pre-term Codes for Errors Detected by Operating System (Continued)

Error	Meaning
92C0	INVALID OPEN <F-N> <ADR> HOST UNWILLING TO BEGIN DIALOG
92D0	INVALID OPEN <F-N> <ADR> INVALID FILE SECURITY
9300	INVALID OPEN <F-N> <ADR> INVALID FILE ID
9310	INVALID OPEN <F-N> <ADR> INVALID HDW / I/O
9320	INVALID OPEN <F-N> <ADR> INVALID LOCK USE
9330	INVALID OPEN <F-N> <ADR> INVALID RECORD SIZE
9340	INVALID OPEN <F-N> <ADR> INVALID REQUESTR
9350	INVALID OPEN <F-N> <ADR> INVALID REVRS
9360	INVALID OPEN <F-N> <ADR> INVALID SEQ O/I
9370	INVALID OPEN <F-N> <ADR> INVALID VAR RECORD SIZE
9380	INVALID OPEN <F-N> <ADR> NO I LABEL
9390	INVALID OPEN <F-N> <ADR> OVRSIZ DISK AREA
9400	INVALID OPEN <F-N> <ADR> OVRSIZ EOF
9410	INVALID OPEN <F-N> <ADR> SMASHED FIB
9420	INVALID OPEN <F-N> <ADR> INVALID BCT PARAMETER
9430	INVALID OPEN <F-N> <ADR> FILE NOT RELEASED BY CLOSE
9440	INVALID OPEN <F-N> <ADR> INVALID LABEL USE ADDRESS
9450	INVALID OPEN <F-N> <ADR> INVALID SHARED RECORD SIZE / BLOCK
9460	<P-N> ABORTED — FAULT IN MCP <FAULT INDICATOR>
9470	ADDRESS ERR <ADR>
9480	INSTRUCTION TIME OUT <ADR>
9490	INVALID BCT ADDRESS
94A0	<P-N> ABORTED — USER FAULT <FAULT INDICATOR>
9500	INVALID INSTRUCTION <ADR>
9510	MEM PAR ADDRESS <ADR> (DS OR DP ONLY - NO ARM)
9520	PROGRAM SNAP ABORTED
9530	INVALID READ <ADR> INVALID FIB ADDRESS
9540	INVALID READ <F-N> <ADR> FILE RSTRCTD OR NOT OPEN
9550	INVALID READ <F-N> <ADR> INVALID BUFFR RSLT DESC
9560	INVALID READ <F-N> <ADR> INVALID DISK ADDRESS
9570	INVALID READ <F-N> <ADR> INVALID I/O OR FILE TYPE
9580	INVALID READ <F-N> <ADR> INVALID RECORD SIZE
9590	INVALID READ <F-N> <ADR> SMASHED FIB
95A0	INVALID READ <F-N> <ADR> SYSTEM ERROR
95B0	INVALID READ <F-N> <ADR> NO USER MEMORY
95C0	<P-N> INVALID DISK ID <F-N> <DISK ID>
9600	INVALID READ <F-N> <ADR> SMASHED KEY
9610	INVALID POSITION <F-N> <ADR> POSITION OUT OF FILE
9620	INVALID READ <F-N> <ADR> FILE NOT SHARED

Table A-19. Pre-term Codes for Errors Detected by Operating System (Continued)

Error	Meaning
9630	INVALID WRITE OR UNLOCK <F-N> <ADR> DISK OR PACK NOT LOCKED
9640	STATEMATE NO LABEL <F-N> <ADRS>
9650	<P-N> DS-ED
9660	INVALID ARM ADDRESS <ADR> (DS OR DP ONLY - NO ARM)
9670	INVALID BCT PARAMETER <ADR>
9680	OVER TIME <ADR>
9690	<P-N> INVALID MIX BCT VARIANT 6 ADDRESS SEG# (MAXJOBS > 99)
96A0	INVALID CLOSE <F-N> <ADR> SYSTEM ERROR
96B0	INVALID CLOSE <F-N> <ADR> NO USER MEMORY
96C0	INVALID CLOSE <F-N> <ADR> INVALID BNA REQUEST
9700	INVALID STOQ <ADR> INVALID BLOCK ADDRESS
9710	INVALID STOQ <ADR> INVALID ENT SIZE
9720	INVALID STOQ <ADR> INVALID NAME
9730	INVALID STOQ <ADR> INVALID NAME SIZE
9750	INVALID MTP LABEL <F-N> <ADR>
9760	MTP LABEL WRITE ERR <F-N> <ADR>
9770	INVALID OPEN EXTEND <F-N> <ADR> INVALID LABEL
9780	INVALID USE RET
9790	INVALID CLOSE <F-N> <ADR> PACK <P.SER.NBR> NOT AVAILABLE
97A0	<P-N> PARITY ERROR ON ROLL-IN
97B0	<P-N> SYSTEM ACCESS SECURITY ERROR — NO USER TABLE ENTRY
97C0	<P-N> INVALID PROGRAM CALL <ADR> PARAMETER FILE ERROR
97D0	<P-N> INVALID PROGRAM CALL <ADR> CALLEE B-O-J ERROR
97E0	<P-N> INVALID PROGRAM CALL <ADR> CALLEE TERMINATION ERROR
9800	INVALID CLOSE <F-N> <ADR> INVALID USER PACK HEADER
9810	INVALID OPEN <F-N> <ADR> OVRSIZE PACK AREA
9820	INVALID READ <F-N> <ADR> INVALID PACK ADDRESS
9830	LOST HANDLER (JOB ABORTED)
9840	INVALID BCT USE
9850	SYS ACCESS ERR INVALID SYS ACCESS CODE
9860	PROG STACK OFLOW
9870	IN USE PACK POWERED OFF
9880	INVALID OPEN <F-N> <ADR> INVALID MCS BUFFER SIZE
9890	INVALID BCT REQUEST - NULL MIX REQUIRED
9900	DMS ERR <ADR> INVALID BCT PARAMETER

Table A-19. Pre-term Codes for Errors Detected by Operating System (Continued)

Error	Meaning
9910	DMS ERR <ADR> NO ERR LABEL
9920	DMS ERR <ADR> DATA BASE NOT OPEN
9930	DMS ERR <ADR> USE PROC CALL FROM USE PROC
9940	DMS ERR <ADR> USER TBL OFLOW
9950	<P-N> DFS VIOLATION
9960	DMS ERR <ADR> DBP TBL OFLOW
9A00	INVALID DIR I/O OPEN <ADR> NOT ALLOWED TO USER
9A10	INVALID DIR I/O OPEN <ADR> INVALID FIB ADDRESS
9A20	INVALID DIR I/O OPEN <ADR> FILE NOT CLOSED
9A30	INVALID DIR I/O OPEN <ADR> BIG FIB REQUIRED
9A40	INVALID DIR I/O OPEN <ADR> SMASHED FIB
9A50	INVALID DIR I/O OPEN <ADR> INVALID UNIT
9A60	INVALID DIR I/O OPEN <ADR> INVALID XD FILE
9A70	INVALID DIR I/O CLOSE <ADR> INVALID FIB ADDRESS
9A80	INVALID DIR I/O CLOSE <ADR> SMASHED FIB
9A90	INVALID DIR I/O CLOSE <ADR> FILE NOT OPEN
9AA0	INVALID DIR I/O REQUEST <ADDRESS> INVALID OP CODE
9AB0	INVALID DIR I/O REQUEST <ADDRESS> I/O IN QUEUE
9AC0	INVALID DIR I/O REQUEST <ADDRESS> CANCEL NOT ALLOWED TO I/O
9AD0	INVALID DIR I/O REQUEST <ADDRESS> CANCEL NOT ALLOWED TO XD FILE
9AE0	INVALID DIR I/O REQUEST <ADDRESS> CANCEL NOT ALLOWED TO DIRECT READ
9B00	INVALID DIR I/O CLOSE <ADR> INHIB NOT ALLOWED
9B10	INVALID DIR I/O REQUEST <ADR> INVALID FIB ADDRESS
9B20	INVALID DIR I/O REQUEST <ADR>
9B30	INVALID DIR I/O REQUEST <ADR> FILE NOT OPEN
9B40	INVALID DIR I/O REQUEST <ADR> INHIB NOT ALLOWED
9B50	INVALID DIR I/O REQUEST <ADR> SMASHED FIB
9B60	INVALID DIR I/O REQUEST <ADR> INVALID ACC CHAN
9B70	INVALID DIR I/O REQUEST <ADR> NOT ALLOWED TO UNIT
9B80	INVALID DIR I/O REQUEST <ADR> INVALID UNIT
9BA0	BNA ERR <ADR> INVALID FIB ADDRESS
9BB0	BNA ERR <ADR> INVALID DATA ADDRESS
9BC0	BNA ERR <ADR> INVALID BUFFER INDEX
9BD0	BNA ERR <ADR> USER NOT PRESENT
9BE0	BNA ERR <ADR> REMOTE DIALOG ABORTED
9BF0	<P-N> SMASHED EVENT DIRECTORY
9C00	INVALID PORT <RQST> <ADR> INVALID FIB ADDRESS

Table A-19. Pre-term Codes for Errors Detected by Operating System (Continued)

Error	Meaning
9C10	INVALID PORT <RQST> <PORT/NUMBER> <ADR> SMASHED FIB
9C20	INVALID PORT <RQST> <PORT/NUMBER> <ADR> INVALID SUB-PORT NUMBER
9C30	INVALID PORT <RQST> <PORT/NUMBER> <ADR> INVALID CODE FILE FMT
9C40	INVALID PORT <RQST> <PORT/NUMBER> <ADR> MISSING OR INVALID ATTRIBUTE
9C50	INVALID PORT <RQST> <PORT/NUMBER> <ADR> INVALID ATTRIBUTE IN FPB
9C60	INVALID PORT <RQST> <PORT/NUMBER> <ADR> INVALID ATTRIBUTE VALUE IN FPB
9C70	INVALID PORT <RQST> <PORT/NUMBER> <ADR> INVALID SUB NUMBER IN FPB
9C80	INVALID PORT <RQST> <PORT/NUMBER> <ADR> INVALID TYPE
9C90	INVALID PORT <RQST> <PORT/NUMBER> <ADR> INVALID SECURITY TYPE
9CA0	INVALID PORT <RQST> <PORT/NUMBER> <ADR> SUBPORT NOT CLOSED
9CB0	INVALID PORT <RQST> <PORT/NUMBER> <ADR> INVALID RECORD SIZE
9CC0	INVALID PORT <RQST> <PORT/NUMBER> <ADR> INVALID RECORD ADDRESS
9CD0	INVALID PORT <RQST> <PORT/NUMBER> <ADR> INVALID FILE STATUS ADDRESS
9CE0	INVALID PORT <RQST> <PORT/NUMBER> <ADR> SUBPORT NOT OPEN
9CF0	INVALID PORT <RQST> <PORT/NUMBER> <ADR> EOF NO LABEL
9D00	INVALID PORT <RQST> <PORT/NUMBER> <ADR> MISSING KEY
9D10	<P-N> LDHST REQUIRED (IPP Error)
9D20	<P-N> HOST MESSAGE REJECTED (IPP Error)
9D30	<P-N> DLP ERROR (IPP Error)
9D40	<P-N> UNEXPECTED TRANSIT TO IDLE (IPP Error)
9D50	<P-N> INVALID VSID MESSAGE (IPP Error)
9D60	<P-N> INVALID BRK USE ROUTINE (Timesharing Error)
9D70	<P-N> NO BRK USE ROUTINE (Timesharing Error)
9D80	<P-N> NO RESULT DESCRIPTOR (Fault Handler Error)
9D90	<P-N> SMASHED MEMORY AREA (Fault Handler Error)
9DA0	<P-N> INVALID TRAP ADDRESS (Fault Handler Error)
9DB0	<P-N> FAIL INSTRUCTION <ADR> (Fault Handler Error)
9DC0	<P-N> DEBUG DS-ED (Fault Handler Error)

LSTMOD (MCP/VS SOURCE LISTING UTILITY)

LSTMOD is a utility program that prints the source code of any desired module or group of modules within the operating system. LSTMOD operates on printer backup files that contain the operating system source code.

MCP/VS source code is provided in two printer backup files named LSAxxx and LSBxxx, where xxx represents the release level of the operating system. LSTMOD cannot print module listings from any printer backup files other than LSAxxx or LSBxxx. Within the printer backup files, each module is identified by a three-letter code.

LSAxxx contains:

- The linker listing (produced when the modules of the operating system are linked together). This listing contains a complete list of modules and their three-letter codes. It should be printed and kept available for easy reference. The linker listing itself uses the code MCP.
- The Module Interface Definition (MID) listing of the operating system. This listing uses the code MID.
- The first group of source code modules in alphabetic order. (For MCP/VS 2.0, the LSA200 file contains the modules ALC through KRN.)

LSBxxx contains:

- The remainder of the source code modules in alphabetic order. (For MCP/VS 2.0, the LSB200 file contains the modules KRS through WTG.)

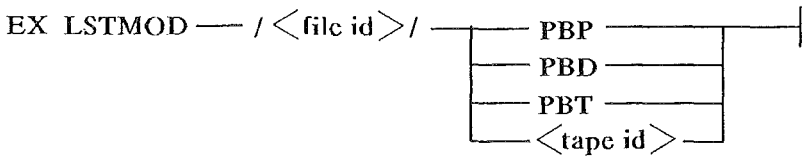
LSTMOD can operate on either LSAxxx or LSBxxx as long as the file is stored in one of the following forms:

Printer Backup Tape File
Printer Backup Pack File on Disk or Diskpack
Printer Backup Disk File on Disk or Diskpack
Printer Backup Pack File on a SYSTEM/COPY Tape
Printer Backup Disk File on a SYSTEM/COPY Tape

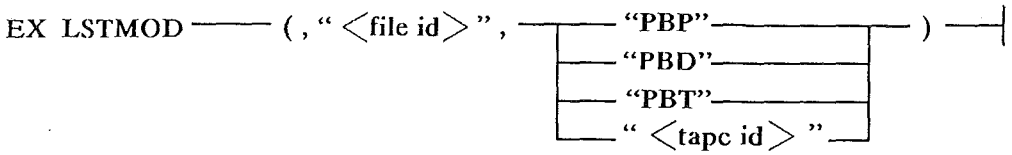
LSTMOD runs most quickly when the printer backup file is stored on disk or diskpack. Directory links are written into the two printer backup files, enabling LSTMOD to search for modules quickly. If the printer backup files are stored on tape (either printer backup tape or a SYSTEM/COPY tape), LSTMOD must search sequentially through the files to locate modules.

Executing LSTMOD

To execute LSTMOD under an operating system prior to MCP/VS 2.0, use this syntax:



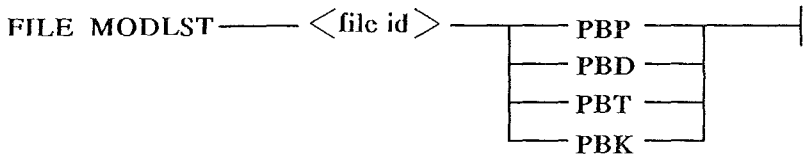
To execute LSTMOD under MCP/VS 2.0 or a later release of the operating system, use this syntax:



The parameters of LSTMOD are used as follows:

- <file id> should be either LSAxxx or LSBxxx, where xxx represents the release level of the operating system. Example: MCP/VS 2.0 = LSA200 and LSB200.
- The second parameter indicates the medium where the source code printer backup file resides.
  - PBP instructs LSTMOD to look for the printer backup file on diskpack, whether it is a printer backup pack file or a printer backup disk file.
  - PBD instructs LSTMOD to look for the printer backup file on disk, whether it is a printer backup pack file or a printer backup disk file.
  - PBT instructs LSTMOD to read a printer backup tape file. Printer backup tape files must contain 136-byte records, blocked 12.
  - If any value other than PBP, PBD or PBT is entered as the second parameter, the value is assumed to be the ID of a SYSTEM/COPY tape.

The output of LSTMOD is directed to a printer by default. To direct the output to another destination, include a file equate statement when LSTMOD is executed. The file equate statement has this syntax:



Example:

EX LSTMOD (,LSA200",PBP) ; FILE MODLST LISTNG PBD

If the printer backup files LSAxxx and LSBxxx are reformatted in any way, the directory links used by LSTMOD become incorrect. If this happens, LSTMOD can be instructed to search for modules sequentially (as it does when the printer backup files are stored on tape). To force a sequential search of the files, execute LSTMOD with a VA 0 1 statement.

Example:

```
EX LSTMOD (,"LSA200","PBP") ; VA 0 1
```

Selecting Modules Through LSTMOD

When LSTMOD is executed, it displays a message on the ODT asking for a list of modules to be printed. Enter the list of desired modules through the AX command. Refer to each module by its three-letter code. Separate the three-letter codes with spaces. A maximum of 100 modules can be printed. LSTMOD continues to accept AX commands until it receives either the 3-character string END, or an AX command containing no characters. The order of the module names does not matter, except that END terminates the list regardless of where it is entered.

For example, to print the modules KTO and PCR from a printer backup file, enter:

```
<Mix No> AX KTO PCR END
```

An AX command of LIBLST END prints the entire backup file.

After the desired modules have been printed or placed in a new printer backup file, the source printer backup file (LSAxxx or LSBxxx) is closed with release. The source printer backup file is *not* purged.

INDEX

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)