Burroughs

Reference Manual

A Series System Software Support

(Relative to the Mark 3.6 System Software Release)

Distribution Code SE

Priced Item
Printed in U.S.A.
December 1985

1170016

Comments or suggestions regarding this document should be submitted on a Field Communication Form (FCF) with the Class specified as "2" (System Software), the Type specified as "1" (F.T.R.), and the Product specified as the seven-digit form number of the manual (for example, "1170016"). The FCF should be sent to the following address:

Burroughs Corporation
PA&S/Mission Viejo
19 Morgan
Irvine, CA 92718

CONTENTS

1 INTRODUCTION

This manual describes some of the important software operations available to users of Burroughs A Series and B 5000/B 6000/B 7000 Series computers. These functions include recording system events, testing or analyzing datacomm facilities, initiating and analyzing memory dumps, and managing system resources.

This manual is a reference manual intended primarily for use by Burroughs systems support personnel and customer software support personnel. The reader is assumed to be familiar with the Burroughs A Series and B 5000/B 6000/B 7000 Series systems.

The notation used in this manual to represent the syntax for the various programs is the "railroad" syntax diagram, which is frequently used in Burroughs manuals. For those unfamiliar with this notation, a description is provided in the "Understanding Railroad Diagrams" section at the end of this manual.

The remainder of this section describes the structure of this manual and the documents that relate to this manual.

STRUCTURE OF THIS MANUAL

Each subject is described in a separate section. The sections are independent of each other and are arranged in alphabetical order.

This manual is divided into the following sections:

1. INTRODUCTION

   This section introduces this manual. Each section of the manual is described and related documents are cited.

2. B 7000 SOFTWARE FEATURES

   This section describes the Failure Analysis (FA) and System Balancing features of B 7000 Series Systems.

3. DCAUDITOR

   This section describes the DCAUDITOR program that performs analysis of an NSPAUDIT file produced by the B 5900/B 6900/B 7900 systems and A Series system datacomm subsystem procedures of the MCP.

4. DCSTATUS

   This section describes the SYSTEM/DCSTATUS utility, which makes use of the DCSYSTEMTABLES installation intrinsic to produce an analysis of the datacomm tables maintained by the MCP and the datacomm subsystem (DCP or NSP).

5. DUMPANALYZER

   This section describes the SYSTEM/DUMPANALYZER utility, which can be invoked with various options to analyze a memory dump.

6. ISAM

   This section describes a set of software routines that implement the indexed sequential access method (ISAM) of storing and retrieving data records.

7. KEYEDIO

   This section describes the SYSTEM/KEYEDIO library, which provides the indexed sequential access method (ISAM) for COBOL74 and RPG. This section also includes descriptions of keyed file structure, coarse tables, fine tables, file management, keyed file attributes, and KEYEDIO procedures.

8. MEMORY DUMP PROCEDURES

   This section describes procedures for forcing memory dumps.

9. MEMORY MANAGEMENT

   This section describes the organization, allocation, and management of memory space on A Series and B 5000/B 6000/B 7000 Series systems.

10. PERIPHERAL TEST DRIVER (PTD)

    This section describes the Peripheral Test Drive (PTD) program that interprets op-codes that are found in test case S-code files created to test peripheral equipment on a system.

11. PRINTBINDINFO

    This section describes the SYSTEM/PRINTBINDINFO utility, which is used to print an analysis of the information used by the Binder when binding a code file.

12. SOFTWARE COMPILATION

    This section describes the Work Flow Language job WFL/COMPILE/SOFTWARE, which is used to compile and install patches in the system software.

13. SUMLOG

    This section describes the system summary log file, SUMLOG, which is used on A Series and B 5000/B 6000/B 7000 Series systems to record information concerning jobs previously run, past MCP activity, and other related information concerning the past status of the machine environment.

14. SWAPPER

    This section describes the SWAPPER facility, which causes tasks running in a time-slicing environment to be swapped in and out of memory according to parameters set by the installation.

In addition an explanation of railroad diagrams and a glossary appear at the end of this manual.

RELATED DOCUMENTS

The following documents contain related information:

Document	Form No.
ALGOL Reference Manual	1169844
Burroughs Network Architecture (BNA), Version 2, Operations Interface, Volume III, Program Agent User's Guide	1188026
COBOL Reference Manual	1169786
COBOL ANSI-74 Reference Manual	1169877
DCALGOL Reference Manual	5014574
DMSII DASDL Reference Manual	1163805
DMSII User Language Interface Software Operation Guide	1180536
FORTRAN Reference Manual	1182441
Mark PTDTESTS Tape	
Operator Display Terminal (ODT) Reference Manual	1169612
PL/I Reference Manual	1169620
Report Program Generator (RPG) Reference Manual	1169760
System Software Site Management Reference Manual	1170008
System Software Utilities Reference Manual	1170024
Work Flow Language (WFL) Reference Manual	1169802

2 B 7000 SOFTWARE FEATURES

This section describes the "Failure Analysis" and "System Balancing" features associated with B 7000 Series systems.

2.1 FAILURE ANALYSIS

The goals of the Failure Analysis (FA) feature of the B 7000 Series system are simplicity in operation and implementation, reliable error and action reporting, and above all, FA actions (such as box removal) that would best preserve system continuity.

2.1.1 FA AND ITS COUNTER MATRIX

FA monitors the condition of the various components of the system primarily through a series of counters that are incremented each time a chargeable hardware error occurs. When an installation determined THRESHOLD is achieved on a component, an attempt is made to eliminate its disruptive influence. The setting of this action THRESHOLD should be carefully determined according to the specific operational considerations and needs of a given installation.

There are four categories of counters:

1.	REQUESTOR (8)	For charges against CPMs and IOMs. There is one per box.
2.	MCM (8)	For charges against MCMs. There is one per MCM.
3.	INTERFACE (64)	For the interface each requestor has to each MCM.
4.	MF2 (8)	For counting one bit errors on an MCM basis.

When a counter's THRESHOLD is exceeded, actions taken vary depending upon counter type and system configuration. Table 2-1 ("FA Threshold Actions") summarizes these actions.

Incidents that draw the attention of FA and cause counter action are itemized in Table 2-2 ("FA Chargeable Errors").

Simply stated, action against a requestor involves the offender being removed without a Halt/Load. Action against an MCM involves a Halt/Load. For interface errors (those involving an MCM and requestor), the total of all interface errors for each of the involved modules is compared, with action being directed at the module with the greatest number of interface errors.

In all cases, FA actions are logged; also, an MCP independent runner named CASUALTYREPORT appears in the mix, displaying the action periodically and requesting acknowledgment. Once acknowledged, CASUALTYREPORT terminates.

CPM CONTROL MODES

If a CPM advances to control mode 2 and the interrupt causing the control mode advance is an acceptable type, the MCP places the CPM back onto the original stack and performs normal error handling and counter action. Failing this criteria, the CPM is removed from the system regardless of its FA counter value. Should this be the only CPM in the system, a Halt/Load will be invoked by way of a fatal dump titled "ONLY CPM IN CM2".

If a CPM should advance to control mode 3, it will delay momentarily upon entry to determine if it is the only CPM that has done so. If this is the case, it will go to a coded "DEAD" stop. System clean-up for this CPM is handled by the remainder of the system when the "DEAD" CPM fails to check in.

All CPMs entering control mode 3 are indicative of a catastrophic memory failure. The action taken here is to determine the failing memory and, if necessary, reconfigure, then perform a Halt/Load to get the system going again. Control mode 3 MCM reconfiguration occurs whenever the MCM containing memory address zero is the memory that has had the major failure. If the entire MCM is lost, then the MCM that contains the control mode 3 logic is reconfigured to fill in for the lost MCM. If an MSU is lost, the MCM is configured to exclude this lost storage unit. The action following reconfiguration is a programmatic Halt/Load. Following this Halt/Load, CASUALTYREPORT informs the user of what action or actions were taken.

MCM FAIL REGISTERS

MCM fail registers are handled by FA whenever they are found to contain error information. Generally, the presence of error information is the result of a memory fail 1 interrupt to an IOM or CPM. To ensure that all MCM failure data is kept current, the registers of all MCMs in the system are polled once every second. When a loaded fail register is found, it is analyzed and the appropriate counter incremented. Table 2-2 ("FA Chargeable Errors") indicates the criteria for counter action.

Single bit errors are logged and counted on an MCM-by-MCM basis for a limit of 25. Once an MCM achieves this limit, one bit errors from it will thereafter be ignored. One bit logging for a given MCM may be restored by using the ODT Primitive ??ZFA (Zero Failure Analysis) command. (Refer to the Operator Display Terminal (ODT) Reference Manual for more information about the use of this command.)

CPM INSTRUCTION RETRY

The B 7000 CPM is capable of performing instruction retry in most cases of processor internal error. It is the responsibility of FA to invoke this feature. Before doing so, FA verifies that certain hardware and software criteria are satisfied. If these criteria are not satisfied, normal error handling is performed.

If the failure reoccurs on the retry, FA again determines that a retry is in order, and if another CPM is available, the next retry is performed by that CPM. Failure to find another CPM or not meeting retry criteria invokes normal error handling.

The criteria for instruction retry on the B 7800 CPM for a PROCESSOR INTERNAL interrupt and a LOOP interrupt are as follows:

a. The CPM is in a normal state.

b. The CPM is not in vectormode.

c. The MES bit in the processor is RESET. MES indicates the processor is in a nonrecoverable state.

For the B 7700 CPM instruction, retry occurs under the same CPM conditions only for a PROCESSOR INTERNAL interrupt.

LOGGING

The mainframe section of the system log receives entries concerning:

a. All CPM alarm interrupts.

b. All IOM errors.

c. All loaded MCM fail registers found (excluding one bits).

d. Limited single bit error entries (as previously described).

e. All FA actions.

These log entries are grouped and presented by the LOGANALYZER as one of four types. (Refer to "Mainframe Error Entry" (MAJOR TYPE=2, Minor Type=12) in the "SUMLOG" section.) These four entry types and their associated information are as shown in Table 2-3 ("Mainframe Log Entries").

FA COMMANDS

Messages used to monitor or set FA parameters are as follows:

LOG MAINFRAME	Causes LOGANALYZER to output mainframe errors.
FAS	Causes the FA counter matrix to be displayed.
??SFL <number>	Sets the action THRESHOLD used by FA.
??ZFA <box type> <box number>	Clears the FA counter for the indicated box and its associated interface counters. If <box type> and <box number> are empty, all FA counters are cleared.

Table 2-1. FA Threshold Actions

COUNTER EXCEEDING THRESHOLD	INVOLVED MODULE IS REDUNDANT	INVOLVED MODULE IS NOT REDUNDANT	H-L REQUIRED
REQUESTOR	Module Removed	Module Removal Aborted	No
MCM	MCM Removed	MCM Removal Aborted	Yes
INTERFACE (Reg Err Total Lss MCM Err Total)	MCM Removed	MCM Removal Aborted	Yes
INTERFACE (Reg Err Total Gtr MCM Err Total)	Requestor Removed	Requestor Removal Aborted	No
MF2 (Count Eql 25)	Ignore Subject MCM's 1-Bit Errs	Ignore Subject MCM's 1-Bit	No

Table 2-2. FA Chargeable Errors

MODULE REPORTING ERROR	TYPE OF ERROR	MODULE CHARGED WITH ERROR	EXCEPTIONS
CPM	Memory Parity	MCM	
CPM	Processor Internal	REQ	First Retry Successful
CPM	Loop	REQ	Interrupted operator is linked list lookup
CPM	Invalid Address	REQ	Referenced address is not in system
IOM	Initiate Busy Channel	REQ	None
IOM	Illegal Home Address	REQ	None
IOM	Buffer Register Parity	REQ	None
IOM	Residue Error	REQ	None
IOM	Address Residue	REQ	None
IOM	Store Disparity	REQ	None
IOM	No Access	REQ	Reference address is not in system
IOM	Fetch Disparity	MCM	None
MCM	Internal	MCM	None
MCM	2 Bit	MCM	
MCM	Control Word Parity	INTERFACE	None
MCM	Illegal Operation	INTERFACE	None
MCM	Wrong Address	INTERFACE	None

Table 2-2. FA Chargeable Errors (cont.)

MODULE REPORTING ERROR	TYPE OF ERROR	MODULE CHARGED WITH ERROR	EXCEPTIONS
MCM	Data Word Parity	INTERFACE	None
MCM	Data Strobe Error	INTERFACE	None
MCM	1-Bit Error	MF2	Limit to 25

Table 2-3. Mainframe Log Entries

MODULE REPORTING ERROR	TYPE NO.	TYPE TITLE AND SUB-HEADINGS	COMMENTS
CPM	1	Type = CPM X Interrupt	"X" is the interrupted CPM's ID
		Cause	The reason the log entry was made
		System Configuration	Mainframe Configuration
		Program Interrupt Point	Where the interrupt occurred. Included are PIR, PSR, SDI, Stack No., Job No., Task No., LL and NCSFF (State).
		Error Report	The interrupt P1, P2 and processor fail register are given in raw and analyzed form.
IOM	2	Type - IOM X Error	"X" is the number of the reporting IOM.
		Cause	Tells how the fail RD was found.
		System Configuration	Mainframe Configuration.
		Error Report	The fail RD is given raw and analyzed form.

Table 2-3. Mainframe Log Entries (cont.)

MODULE REPORTING ERROR	TYPE NO.	TYPE TITLE AND SUB-HEADINGS	COMMENTS
MCM	3	Type = MCM X Error	"X" is the reporting MCM.
		Cause	How the fail register was located.
		System Configuration	Mainframe configuration.
		Error Report	The fail register is given in raw and analyzed form.
MCP	4	Type = System Error Action	FA is reporting some action it performed.
		Cause	What FA did (or could not do).
		Reason	Why the action was initiated.
		System Configuration	Mainframe configuration following action.
MCP	5	Type = Mainframe Event	FA is reporting a system event.
		Cause	Event being recorded (dump or Halt/Load).
		System Configuration	Mainframe configuration.
		Report	Halt/Load or dump reason. For dumps, includes calling stack number, job number, and task number and whether fatal or non-fatal.

Table 2-3. Mainframe Log Entries (cont.)

MODULE REPORTING ERROR	TYPE NO.	TYPE TITLE AND SUB-HEADINGS	COMMENTS
MCP	6	Type = System Reconfiguration Action	FA is reporting a voluntary change in mainframe configuration.
		Cause	Box involved and whether SaVed, ReadYed or SWAPped (if an MCM).
		System Configuration	Mainframe configuration following action.
		Report	Whether change was operator requested or the aftermath of on-line CPM testing.

2.2 SYSTEM BALANCING

SYSTEM BALANCING is a feature of the MCP that gives the user a set of functions by which he can "balance" or "tune" his system to a set of operations. In general, these functions provide the user with the ability to monitor the overall system utilization and the ability to change various software parameters that affect system utilization.

SYSTEM MONITORING

Mechanisms have been installed in the MCP to gather statistics that define SYSTEM UTILIZATION in two categories, utilization of processing and utilization of input/output, over a user-defined time interval. The description and purpose of the system utilization components follow.

UTILIZATION OF PROCESSING

The utilization of processing is subdivided, into four components, each described in percentage of the last time interval.

a. Usertime - the time logged directly to user jobs (less the time spent for handling physical I/O). User time includes all user jobs and "visible" MCP independent runners, such as library maintenance.

b. I/O time - the percentage of processing time over the last interval that is spent handling physical I/O functions. This component is critical to system utilization because the amount of I/O activity is directly dependent on the amount of processing given to the I/O subsystem. The user can observe this component and vary certain "system balancing parameters" to achieve a desired amount of I/O activity.

c. MCP time - the percentage of time over the last interval spent by the operating system (MCP) to "manage" the system.

d. Idle time - the percentage of time that the processor(s) was (were) not used for user, I/O or MCP functions above.

UTILIZATION OF INPUT/OUTPUT

The utilization of I/O is subdivided into several components, each described in rates calculated over the last time interval.

a. User I/O - the average number of user I/O(s) per second and the number of kilobytes of data transferred per second by user I/O(s).

b. MCP I/O - this component is expressed in the same terms as user I/O, but applies to I/O(s) performed by the MCP. MCP I/O(s) include such things as overlays and logging.

c. Total I/O - the total system I/O activity; the sum of user I/O(s) and MCP I/O(s).

d. I/O interrupts - the average number of I/O interrupts per second calculated over the time interval.

The system utilization can be displayed on the console via invocation of the ODT U (Utilization) command. This option is also available in the Automatic Display Mode (ADM) ODT display. An example of this ADM option is as follows:

```
ADM (U 7, MSG) DELAY 10.
```

In this case, the "SYSTEM UTILIZATION" followed by "MESSAGES" would be displayed on the console every ten seconds. An example of the output of the utilization message is as follows:

```
----- SYSTEM UTILIZATION -----
CPM: USER = 68%       MCP = 5%
     I/O  = 10%       IDLE = 17%
I/O: USER = 60    IO/SEC (220 KB/SEC)
     MCP  = 12    IO/SEC ( 40 KB/SEC)
     TOTAL= 72    IO?SEC (260 KB/SEC)
     43 IO-INTERRUPT/SEC
```

IMPLEMENTATION

The system utilization statistics are kept within the MCP. Since the nature of the statistics require that they be placed in the main stream of the MCP, a slight overhead is encountered.

Because these statistics are only meaningful if considered over a time interval, the MCP resets the statistic counters at the end of every interval. The MCP procedure ONESECONDBURDEN, which is called approximately once every second, performs this resetting action. Therefore, the time interval is at least one second and is an integral multiple of seconds. The system utilization statistics are stored in two arrays. One array contains the counts from the previous time interval. The other array contains the counts from the end of the previous time interval to the present.

A global MCP procedure "SYSTEMSTATISTICS" is responsible for retrieving these statistics. When called, this procedure calculates an interval's worth of values by an interpolation of the previous time interval and the current time interval values. This method is used to allow the statistics to be interrogated at any point during the time interval. Procedure SYSTEMSTATISTICS has one parameter, an array. It is a real procedure that returns a value of zero if no errors were encountered and a one if the input array is too short. The procedure returns the following statistics in the array:

WORD #	DESCRIPTION
1	Percentage of idle time.
2	Percentage of I-O time.
3	Percentage of user time.
4	Percentage of MCP time.
5	User I-O(s)-per-second.
6	User I-O kilobytes-per-second.
7	MCP I-O(s)-per-second
8	MCP I-O kilobytes-per-second.
9	Total I-O(s)-per-second.
10	Total I-O kilobytes-per-second.
11	I-O interrupts-per-second.

A user program, written in DCALGOL, can obtain these system utilization statistics by calling GETSTATUS, using type 2 (MISCREQUEST) and subtype 3. This subtype simply causes the CONTROLLER to make a call on the MCP procedure SYSTEMSTATISTICS. The system utilization statistics in the format just described are returned in the array passed to GETSTATUS. The following call would obtain the current utilization statistics:

```
B := GETSTATUS (0 & 2 GSTYPEF & 3 GSSUBTYPEF, 0, 0, A);
```

Where "B" is a Boolean, "GSTYPEF" is the field 7:8, "GSSUBTYPEF" is the field 15:8 and "A" is an array. Before the call on GETSTATUS, A[0] must contain the value 2.

DYNAMIC VARIATION OF SYSTEM-BALANCING PARAMETERS

"Dynamic Variation of System-Balancing Parameters" is the ability to change, while the MCP is running, certain software parameters that control the flow of the MCP. In particular, the user is given the ability to specify the MCP's I/O interrupt scheme and the ability to change the system utilization time interval. Both of these parameters can be altered or interrogated by using the ODT SBP (System Balancing Parameters) command. (Refer to the Operator Display Terminal (ODT) Reference Manual for more information about the use of this command.)

The keyword "SBP" may be entered followed by one of two parameters: "INTERVAL" (used to change the system utilization time interval) and "IOINTERRUPT" (used to change the MCP I/O interrupt schemes). "SBP" with no parameter following it causes the current values of INTERVAL and IOINTERRUPT to be displayed. The INTERVAL is expressed in terms of seconds and IOINTERRUPT is a choice of four interrupt strategies. The SBP command may take one of the following forms:

```
SBP (request display of current balancing parameters)
SBP INTERVAL <number> (for example, SBP INTERVAL 300)
SBP IOINTERRUPT QEMPTY
SBP IOINTERRUPT IOFINISH
SBP IOINTERRUPT IDLE
SBP IOINTERRUPT WAITING
SBP IOINTERRUPT WAITING QEMPTY
```

INTERVAL PARAMETER

INTERVAL is the system utilization time interval as discussed under "System Monitoring". A small interval gives an immediate picture of system utilization, and a large interval gives a more overall, averaged picture. With a small interval, radical changes are accurately reflected. With a large interval, radical changes are "smoothed out". The default interval is ten seconds, which can be termed a small interval. The user may desire a larger interval for an overall picture, but a smaller one is not recommended because the statistics tend to fluctuate too much.

See also
    System Monitoring 17

IOINTERRUPT PARAMETER

IOINTERRUPT is the system balancing feature that allows a choice of four I/O interrupt strategies. The differences among the four involve the way various types of jobs interact and include the amount of CPM time devoted to handling I/O completes. The four interrupt strategies are as follows:

a. QEMPTY - Specifies that the IOM(s) interrupt a CPM when the last I/O request for any unit is completed and on every I/O completion if a CPM is idle.

b. IOFINISH - Specifies that IOM(s) interrupt a CPM upon handling every I/O completion. This causes I/O bound jobs to run in a shorter elapsed time, but requires more CPM time to handle I/O completions.

c. IDLE - Provides for no I/O interrupts except when a CPM is idle. The effect of this is to bias CPM usage toward CPM-bound jobs. This requires the least amount of CPM time for handling I/O completions.

d. WAITING - Specifies that I/O interrupts occur upon completion of an I/O operation only if a process is waiting for that I/O or if a CPM is idle. As long as the progress of no process is dependent on the completion of a particular I/O operation, the IOM generates no interrupt. The effect of this strategy is to use just enough CPM time to keep I/O bound jobs running.

This WAITING interrupt strategy may be specified along with QEMPTY by using the following form of the ODT SBP command:

```
SBP IOINTERRUPT WAITING QEMPTY
```

IOINTERRUPT WAITING QEMPTY is the default strategy when none is specified.

IMPLEMENTATION

The SETSTATUS procedure of the MCP handles the SBP command. Within SETSTATUS, MCP global variables are altered to change the interval or IOINTERRUPT strategy. If the user wishes to change the system balancing parameters from a user program, he can use the DCKEYIN function of DCALGOL, or he can invoke SETSTATUS, in which case he should refer to procedure "SYSTEMBALANCINGPARAMETERS" in the CONTROLLER for specific details of input formats, and so forth.

SYSTEM-BALANCING USAGE

System balancing can be used in a number of ways. By observing the system utilization and changing the system-balancing parameters, the user can "tune" the system to help achieve maximum throughput. He can "bias" the system toward a specific type of application; for instance, IOINTERRUPT on idle biases the system towards process-bound jobs and IOINTERRUPT on every I/O FINISH biases the system towards IO-bound jobs. Another use is to record and graph utilization throughout the day, enabling the user to distribute the "system load" on an equitable basis. In general, "system balancing" should enhance the user's knowledge and control of his system.

3 DCAUDITOR

DCAUDITOR is a program that performs analysis of an NSPAUDIT file produced by the B 5900/B 6900/B 7900 systems and A Series systems datacomm subsystem procedures of the Master Control Program (MCP).

The user can identify the items to be audited by using the ID (Initialize Datacomm) Operator Display Terminal (ODT) command audit options. If the audit options are set while a Network Support Processor (NSP) is initializing, the MCP audits these requests and creates an audit file titled "NSPAUDIT/DCINITIAL/<NSP-unitid>". If the audit options are set after an NSP initializes, the MCP audits these requests and creates an audit file titled "NSPAUDIT/DCCONTROL/<NSP-unitid>".

DCAUDITOR performs detailed analysis for NSP requests and results; however, only the TYPE/CLASS field is analyzed for DCP and DCWRITE formatted request and results.

DCAUDITOR RUN STATEMENT

The run statement of DCAUDITOR is as follows:

```
    RUN *SYSTEM/DCAUDITOR("<DCAUDITOR options>"); VALUE = <nnn>
```

where <DCAUDITOR options> is a string of options separated by spaces and <nnn> is the unit number of the NSP whose audit file is to be analyzed.

OPTIONS

<DCAUDITOR options>

```
    ---------------------------|
       |                     |
       |- DCINITIAL ---------|
       |                     |
       |- DCCONTROL ---------|
       |                     |
       |-<backspace count>---|
       |                     |
       |- LINES --<range>----|
       |                     |
       |- LSNS --<range>-----|
       |                     |
       |- STATIONS --<range>-|
```

<backspace count>

```
    --<integer>--|
```

<range>

```
    --<integer-1>------------------|
                  |             |
                  |-<integer-2>-|
```

Semantics:

DCINITIAL
DCCONTROL

The DCINITIAL and DCCONTROL options specify which file is to be analyzed. DCINITIAL selects the NSPAUDIT file created during NSP initialization, and DCCONTROL selects the NSPAUDIT file created after the NSP has initialized.

If these options are not specified, the file that is label-equated to SCAUDITF is analyzed.

<backspace count>

The <backspace count> option restricts the analysis to <backspace count> records of the NSPAUDIT file. Valid integers are 1 to 1048575.

LINES

The LINES option allows the selective analysis of NSP requests and results that pertain to a range of line numbers. A line number is assigned by the Network Definition Language II (NDLII) compiler and is the ordinal number of the line in the SOURCENDLII, starting at 1.

If this option is used, no DCWRITE requests and results are displayed.

LSNS

The LSNS option allows the selective analysis of NSP requests and results that pertain to a range of logical station numbers. A logical station number is assigned by the MCP and is the ordinal number of the station in the SOURCENDLII, starting at 2.

If this option is used, no DCWRITE requests and results are displayed.

STATIONS

The STATIONS option allows a selective analysis of NSP requests and results that pertain to a range of station numbers. A station number is assigned by the NDLII compiler and is the ordinal number of the station in the SOURCENDLII, starting at 1.

If this option is used, no DCWRITE requests and results are displayed.

SAMPLE REPORT

Messages for DCWRITE formatted requests and results are printed in hex. An example appears in the sample report that follows.

```
****************************************************************************************************
##### SCREQUEST #####      IOLENGTH = 7 BYTES      TIMESTAMP = 11:13:40.73
 REQUEST NUMBER  (DOUBLE) VALUE = 1075380317(4C190C5D)
 REQUEST TYPE  (ENUMERATION) VALUE = 22(16): MAKE STATION READY
 STATION NUMBER  (INTEGER) VALUE = 24(0018)
****************************************************************************************************
##### SCRESULT #####      IOLENGTH = 10 BYTES      TIMESTAMP = 11:13:40.75
 RESULT COUNT  (INTEGER) VALUE = 10(000A)
 RESULT NUMBER  (INTEGER) VALUE = 232(00E8)
 RESULT TYPE  (ENUMERATION) VALUE = 21(15): ACKNOWLEDGE REQUEST
 REQUEST NUMBER  (DOUBLE) VALUE = 2147549275(8C010C5B)
****************************************************************************************************
##### DCP RESULT #####      IOLENGTH = 8 WORDS      TIMESTAMP = 11:13:40.82
 DCP MESSAGE TYPE = MAKE LINE READY
        060000800100 000000000000 000000000000 000000000000 000000600000 000000000000 000000000002 000000800100
****************************************************************************************************
##### SCRESULT #####      IOLENGTH = 10 BYTES      TIMESTAMP = 11:13:40.85
 RESULT COUNT  (INTEGER) VALUE = 10(000A)
 RESULT NUMBER  (INTEGER) VALUE = 233(00E9)
 RESULT TYPE  (ENUMERATION) VALUE = 21(15): ACKNOWLEDGE REQUEST
 REQUEST NUMBER  (DOUBLE) VALUE = 1075380316(4C190C5C)
****************************************************************************************************
##### SCRESULT #####      IOLENGTH = 12 BYTES      TIMESTAMP = 11:13:40.87
 RESULT COUNT  (INTEGER) VALUE = 12(000C)
 RESULT NUMBER  (INTEGER) VALUE = 234(00EA)
 RESULT TYPE  (ENUMERATION) VALUE = 1(01): CLEARED STATION
 REQUEST NUMBER  (DOUBLE) VALUE = 0(00000000)
 STATION NUMBER  (INTEGER) VALUE = 24(0018)
****************************************************************************************************
##### SCRESULT #####      IOLENGTH = 12 BYTES      TIMESTAMP = 11:13:40.88
 RESULT COUNT  (INTEGER) VALUE = 12(000C)
 RESULT NUMBER  (INTEGER) VALUE = 235(00EB)
 RESULT TYPE  (ENUMERATION) VALUE = 8(08): STATION NOT READY
 REQUEST NUMBER  (DOUBLE) VALUE = 0(00000000)
 STATION NUMBER  (INTEGER) VALUE = 24(0018)
****************************************************************************************************
##### DCP RESULT #####      IOLENGTH = 8 WORDS      TIMESTAMP = 11:13:40.90
 DCP MESSAGE TYPE = RECALL OUTPUT
        1A0001800100 000000000000 001C3831C383 000000000000 000000290001 000000000000 000000000000 000000000000
****************************************************************************************************
##### DCWRITE RESULT #####      IOLENGTH = 8 WORDS      TIMESTAMP = 11:13:40.92
 DCWRITE MESSAGE CLASS = GOOD RESULTS
        050001000019 000000000000 001C3831C383 2501A7000000 000000290001 000000000000 000000000000 000000000000
****************************************************************************************************
##### DCWRITE RESULT #####      IOLENGTH = 10 WORDS      TIMESTAMP = 11:13:40.93
 DCWRITE MESSAGE CLASS = RECALLED MESSAGE
        060000000019 000000000000 0A00013C0000 2501A8000C00 000000210000 000000000000 002578C2F5F9 FCFC7AF5F5FC FC4CC3C1D5C4
        C50000000000 000016000000 000400C00000 1415001A2000 000000000000 000058B2E3C28 282126000000 000000000000 5D242A29385E
****************************************************************************************************
##### SCRESULT #####      IOLENGTH = 10 BYTES      TIMESTAMP = 11:13:40.95
 RESULT COUNT  (INTEGER) VALUE = 10(000A)
 RESULT NUMBER  (INTEGER) VALUE = 236(00EC)
 RESULT TYPE  (ENUMERATION) VALUE = 21(15): ACKNOWLEDGE REQUEST
 REQUEST NUMBER  (DOUBLE) VALUE = 1075380317(4C190C5D)
****************************************************************************************************
##### DCP RESULT #####      IOLENGTH = 8 WORDS      TIMESTAMP = 11:13:40.97
 DCP MESSAGE TYPE = SET/RESET STATION READY
        0E0001800100 000000000000 000000000000 000000000000 000000250001 000000000000 000000000000 000000000000
****************************************************************************************************
```

4 DCSTATUS

SYSTEM/DCSTATUS is a DCALGOL program that makes use of the DCSYSTEMTABLES installation intrinsic to produce run-time "snapshots" of the datacomm tables maintained by the Master Control Program (MCP) and the datacomm subsystem. (For further information about the DCSYSTEMTABLES installation intrinsic, refer to the DCALGOL Reference Manual.)

DCSTATUS analyzes elements of the datacomm subsystem for Data Communications Processor (DCP) systems (B 6800, B 7700, and B 7800 systems), Message-Level Interface Processor (MLIP) systems (B 5900, B 6900, A 3, A 9 and A 10 systems), and Host Data Unit (HDU) systems (B 7900 and A 15 systems). The DCSTATUS options ALL, STATION, TERMINAL, TABLES, GRAPH, NETWORK, and FILE apply to both DCP and MLIP systems. The options DCP, CLUSTER, LINE, and MODEM apply only to DCP systems. The options NSP and LSP apply only to MLIP and HDU systems.

No attempt is made in this section to interpret the results generated by the DCSTATUS program, because understanding these results requires an understanding of Network Definition Language (NDL) or Network Definition Language II (NDLII), as well as a general familiarity with the DCP or Network Support Processor (NSP). NDL applies to DCP systems and NDLII applies to MLIP and HDU systems.

4.1 EXECUTION

The DCSTATUS program can be invoked as follows:

a. Through the CANDE DCSTATUS command.

b. Through the DIAGNOSTICMCS DP command.

c. Through a CANDE or WFL RUN statement.

CANDE DCSTATUS COMMAND

The CANDE DCSTATUS command may be entered from a remote terminal to execute DCSTATUS and produce a run-time analysis of the current state of the datacomm subsystem.

<CANDE dcstatus command>

```
-- DCSTATUS ------------------------------------------------|
            |                       | |                    |
            |-<dcstatus option list>-| | |<----------------| |
                                      | |                 | |
                                      |--- ; --<modifier>---|
```

Semantics:

<dcstatus option list>

The <dcstatus option list> must consist of a string of standard options allowed by SYSTEM/DCSTATUS. If <dcstatus option list> is not specified, the default is the STATION option with the <lsn> specification set to the user's logical station number (LSN). The output is directed to the user's terminal. (Refer to "DCSTATUS Options" for a complete description of each option.)

NOTE

The ALL, DCP, NSP, and CLUSTER options produce voluminous output.

<modifier>

For a definition of <modifier>, refer to the CANDE Reference Manual.

Examples:

```
DC
#RUNNING 3854
STATION 13
SYSTEM/DCSTATUS (III.0.90)  DATE : 04/20/78  TIME : 1540:08
FILES: ELMONTENDL/NIF, ELMONTENDL/DCPCODE
DATE OF NDL COMPILATION : 03/03/78 , TIME : 0724:44
NO RECONFIGURATION EXECUTED.
STATION 13
STATION NAME = DOCUMENT1
TERMINAL NAME = TD820
DCC STATION TABLE
ENABLED : READY : ATTACHED :
MCS = 1: <lsn> = 13 : WIDTH = 80
PRIMARY Q = 8,CURRENT Q = 8,STN Q = 0
DLS : 0,2,2 ATTACHED TO FILE 1 REL STN NO = 1
NORMAL TERMINATION
#

DC CLUSTER 4,2
#RUNNING 3966
CLUSTER 4,2
SYSTEM/DCSTATUS (III.0.90)  DATE : 04/20/78  TIME : 1555:50
FILES: ELMONTENDL/NIF ELMONTENDL/DCPCODE
DATE OF NDL COMPILATION : 03/03/78 , TIME : 0724:44
NO RECONFIGURATION EXECUTED.
*--DCP IS NOT NDL DEFINED --- 4 ---
NORMAL TERMINATION
#
DCSTATUS GRAPH;FILE LINE(KIND=PRINTER)
#RUNNING 40000
#
```

See also
    DCSTATUS Options. 38

DIAGNOSTICMCS DP COMMAND

The DIAGNOSTICMCS DP command may be entered from a remote terminal to initiate SYSTEM/DCSTATUS and produce a run-time analysis of the current state of the datacomm subsystem. For information beyond that given here, refer to the DIAGNOSTICMCS Reference Manual.

<diagnosticmcs dp command>

```
-- ? -- DP --------------------------- REMOTE --------------------->
          |                      | |          |
          |- ON --<family name>-| |- SITE ---|

  |<-------- ; -------|
  |                   |
>---<dcstatus option>-----------------------------------------------|
```

Semantics:

ON <family name>

The ON <family name> option may be used to specify the family on which SYSTEM/DCSTATUS resides.

REMOTE

Specifying REMOTE causes DCSTATUS output to be sent to the remote station initiating the DP.

SITE

Specifying SITE causes DCSTATUS output to be directed to the printer.

<dcstatus option>

Refer to "DCSTATUS Options" for a complete description of each option.

See also
DCSTATUS Options. 38

Examples:

```
?DP REMOTE(STATION 5)
## OK ##
## FILE OPEN ##

STATION 5

SYSTEM/DCSTATUS (II.8.60) DATE : 12/16/76 TIME : 0623:28
FILES: PACKNDL8/NIF ON PACK, PACKNDL8/DCPCODE ON PACK
DATE OF NDL COMPILATION : 12/12/76 , TIME : 0442:59
NO RECONFIGURATION EXECUTED.

STATION 5

STATION NAME = TTY3.
TERMINAL NAME = TELETYPE.
DCC STATION TABLE
LOGOIN : SEQ MODE : ENABLED : READY : ATTACHED :
MCS = 3: LSN = 5 : WIDTH = 72
STATION REMOTE TYPE = 0: RETRY COUNT = 15: NIF INDEX = 267
CONTROL TO CQ:
PRIMARY Q = 33,CURRENT Q = 33,STN Q = 0
TRANSFERRED : DLS : 0,12,0 ATTACHED TO FILE 1 REL STN NO = 1

NORMAL TERMINATION

## FILE CLOSE ##
```

CANDE AND WFL RUN STATEMENTS

The following CANDE RUN statement may be entered from a remote terminal to initiate SYSTEM/DCSTATUS:

```
RUN *SYSTEM/DCSTATUS ("<dcstatus option list>")

                          or

EXECUTE *SYSTEM/DCSTATUS ("<dcstatus option list>")
```

The CANDE RUN statement, including the <dcstatus option list>, must be in capital letters.

The following WFL job deck may be entered from an operator display terminal (ODT) to initiate SYSTEM/DCSTATUS:

```
<i>  BEGIN JOB
     RUN *SYSTEM/DCSTATUS ("<dcstatus option list>")
<i>  END JOB
```

Refer to "DCSTATUS Options" for a complete description of each allowable DCSTATUS option.

[<task id>] and <task equation list> specifications may be added to the WFL RUN statement as desired. (Refer to the Work Flow Language (WFL) Reference Manual for more information about the RUN statement.)

When either the CANDE or WFL RUN statement is used, DCSTATUS output is sent to the line printer unless LINE is file-equated to KIND=REMOTE.

See also
    DCSTATUS Options. 38

<u>4.2</u> <u>DCSTATUS OPTIONS</u>

The <dcstatus option list> specifies those elements of the datacomm subsystem that are to be analyzed. These options are shown in the following diagram in this subsection. The following options are arranged hierarchically so that the earlier elements listed include all those that follow: ALL, DCP/NSP, CLUSTER/LSP, LINE, STATION. In other words, each higher-order item in the hierarchy (they are listed from highest to lowest) is inclusive of all lower-order items. For example, if CLUSTER is specified, the analysis is performed on all lines and stations on that cluster. However, the options TERMINAL, TABLES, MODEM, NETWORK, GRAPH, and FILE do not fit into this hierarchy. A full explanation of each option is given in the semantics that follow the syntax for <dcstatus option list>.

The internal file name for the output file is LINE. When the WFL RUN statement or the CANDE RUN statement is used to initiate DCSTATUS, output is sent to the line printer by default. When the CANDE DCSTATUS command or the DIAGNOSTICMCS DP command is used, the output is sent to a remote terminal by default. The output format is modified to fit a 72-character line width. The file LINE can be file-equated in any of the above cases if an output device different from the default is wished.

The DCSYSTEMTABLES intrinsic does not lock the various tables that it accesses. Therefore, the contents of the tables may change while the intrinsic is accessing them. In addition, more than one call on the DCSYSTEMTABLES intrinsic is made to obtain the contents of all the various datacomm tables. If the datacomm tables maintained by the MCP change between calls, the results produced by DCSTATUS may appear internally inconsistent. In particular, DCSTATUS results may be incorrect if the current Network Information File (NIF) or DCPCODE file, or both, has been replaced by a newer version with the same file titles. When this situation occurs, DCSTATUS has access only to these new files, and the information in them may not correspond to that contained in the MCP tables being analyzed.

By using the FILE statement, analysis can be performed on Network Definition Language (NDL/NDLII) files other than the currently active network files. Analysis of inactive network files precludes reporting information requiring active network files. The following are the only allowable options for inactive network files:

```
DCP <DCP number> NDL or DCP NDL
NSP <unit id> NDL or NSP NDL
STATION <station number> NDL or STATION NDL
TERMINAL
MODEM
GRAPH
NETWORK
```

DCSTATUS

```
<dcstatus option>

    ---- ALL -----------------------------------------------|
      |                                                  |
      |- DCP ---<DCP number>-----------------------------|
      |       |                                          |
      |       |---------------- NDL ---------------------|
      |       |              |                           |
      |       |-<DCP number>-|                           |
      |                                                  |
      |- NSP ---<unit id>--------------------------------|
      |       |                                          |
      |       |------------- NDL ------------------------|
      |       |           |                              |
      |       |-<unit id>-|                              |
      |                                                  |
      |- CLUSTER --<DCP number>--- , ---<cluster number>-|
      |                         |     |                  |
      |                         |- : -|                  |
      |                                                  |
      |- LSP --<unit id>---------------------------------|
      |                  |                               |
      |                  |- , ---<adaptor number>--------|
      |                  |     |                         |
      |                  |- : -|                         |
      |                                                  |
      |-<line specifications>----------------------------|
      |                                                  |
      |- STATION ----------------------------------------|
      |           |       | |                            |
      |           |-<lsn>-| |- NDL ----------------------|
      |                                                  |
      |- MODEM ------------------------------------------|
      |         |                                        |
      |         |-<modem number>-------------------------|
      |                                                  |
      |- TERMINAL ---------------------------------------|
      |            |                                     |
      |            |-<remote type index>-----------------|
      |                                                  |
      |- TABLES -----------------------------------------|
      |                                                  |
      |- GRAPH ------------------------------------------|
      |         |                                        |
      |         |-<DC file prefix>-----------------------|
      |                                                  |
      |- NETWORK ----------------------------------------|
      |           |                                      |
      |           |-<DC file prefix>---------------------|
      |                                                  |
      |- FILE -------------------------------------------|
               |                                         |
               |-<DC file prefix>------------------------|
```

<line specifications>

```
    -- LINE --<DCP number>---  , --------------------------------------->
                             |     |
                             |- : -|

    >---<line number>--------------------------------------------------|
      |                                            |
      |-<cluster number>---  , ---<adaptor number>-|
                           |     |
                           |- : -|
```

Where <line number> is <cluster number> * 16 + <adaptor number> and <adaptor number> is the number of the line's adaptor.

<DC file prefix>

```
    --<identifier>--------------------------|
                   |                      |
                   |- ON --<family name>-|
```

Semantics:

ALL

Produces a complete analysis of the data comm network. Analysis of all DCP/NSP, cluster/LSP, line and station tables, together with an analysis of the NDL/NDLII for each station and terminal, is performed.

DCP

Produces an analysis of clusters, lines, and stations on all DCPs or on a specific DCP. Use of the NDL option causes reporting to be based on the information contained in the Network Information File (NIF) and DCPCODE files instead of the current datacomm tables. The DCP option applies only to DCP systems.

NSP

Produces an analysis of Line Support Processors (LSPs), lines, and stations on all Network Support Processors (NSPs), or on a specific NSP. Use of the NDL option causes reporting to be based on the information contained in the NIFII and DCPCODE files instead of that contained in the current datacomm tables. The NSP option applies only to MLIP systems.

CLUSTER

Produces an analysis of the lines and stations on the designated cluster. The Cluster option applies only to DCP systems.

LSP

Produces an analysis of the lines and stations on the designated Line Support Processor (LSP). The <adaptor number> option produces an analysis of the one designated line and its stations. The LSP option applies only to MLIP systems.

<line specifications>

Produces an analysis of the designated line and its stations. The <line specifications> option applies only to DCP systems.

MODEM

Produces an analysis of modem information for a specific modem or for all modems defined in the network. The MODEM option applies only to DCP systems.

STATION

Produces a station analysis. If no <lsn> is specified, all stations are analyzed. The normal sources of information for the STATION option are the data comm tables in main memory or those in DCP or NSP local memory. If the NDL option is specified, the sources of information are the NIF/NIFII and DCPCODE files.

TERMINAL

Produces a listing of the NDL specifications of all terminals or of the designated terminal. The <remote type index> is the index used by the MCP to index into a table that describes each terminal specified in the NDL/NDLII. Terminals are numbered in the sequence in which they appear in the NDL/NDLII terminal definitions.

TABLES

Produces a raw hexadecimal dump of the Data Communications Controller (DCC) tables and the DCP/NSP line and station tables.

GRAPH

Produces a graph of the data comm network showing the relationship between the DCPs/NSPs, clusters/LSPs, lines (names, addresses, and phone numbers for dial-in lines), and stations (names and LSNs). Because the graph information is obtained from the network definition files specified by the <DC file prefix>, the GRAPH option can be used whether data comm is running or not. If the <DC file prefix> is not specified, the one currently being used by the system is GRAPHed.

NETWORK

Produces a brief tabular network configuration report. Information in the report includes DCP/NSP, cluster/LSP, line/adaptor, station, terminal, and MCS data. Because the network information is obtained from the network definition files specified by the <DC file prefix>, the NETWORK option can be used whether data comm is running or not. If the <DC file prefix> is not specified, the one currently being used by the system is analyzed.

FILE

When the <DC file prefix> is specified, limited analysis can be performed on any nonactive network definition file. If the <DC file prefix> is not specified, the one currently being used by the system is used. For example, if DCSTATUS is supplied with the following <dcstatus option list>:

```
("FILE A/B; NETWORK; GRAPH; FILE; NETWORK; GRAPH")
```

the first NETWORK and GRAPH reports are generated using A/B/NIF and A/B/DCPCODE; the remaining reports use the NIF and DCPCODE files that are currently being used by the system.

Examples:

The following example shows a portion of what DCSTATUS would produce on the line printer using the NETWORK option for the B5900. A table of information about the network is given that includes the headings NSP#, LSP#, LINE#, STATION type, LSN#, adaptor#, Receive Address (RA), Transmit Address (TA), terminal type, synchronous or asynchronous mode, bits-per-second transmitted, class declared for the terminal in the NDL, and the type of Message Control System (MCS) in use..

```
                    ***********************************************************
                    *        D A T A C O M M   N E T W O R K   S U M M A R Y  *
                    *                                                         *
                    *           FILES  ER69/CJ/NIF ON DISK                    *
                    *                  ER69/CJ/DCPCODE ON DISK                *
                    *                                                         *
                    *           DATE   11/15/83          TIME   1439 07       *
                    ***********************************************************

NSP LSP AD D  L    S  LINE STATION          LSN  RA TA TERMINAL            MODE BPS  CLASS             MCS
************************************************************************************************************
109 114 00 0 000 000 0022 MT133939          0030 39 39 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 01 0 001 000 0023 TD2813G1          0032 G1 G1 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 01 0 001 001 0023 AP281RP           0033 RP RP AP300TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 01 0 001 002 0023 TD2813G2          0034 G2 G2 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 01 0 001 003 0023 MT283950          0035 50 50 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 01 0 001 004 0023 TD286330          0036 30 30 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 01 0 001 005 0023 TD289329          0037 29 29 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 01 0 001 006 0023 TD276310          0031 10 10 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 02 0 002 000 0024 TD576B345         0038 45 45 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 02 0 002 001 0024 TD576B308         0039 08 08 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 02 0 002 002 0024 TD576B343         0040 43 43 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 02 0 002 003 0024 TD576B3D2         0041 D2 D2 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 02 0 002 004 0024 TD576B3D3         0042 D3 D3 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 02 0 002 005 0024 TD576B346         0043 46 46 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 02 0 002 006 0024 TD576B347         0044 47 47 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 03 0 003 000 0025 TD294314          0045 14 14 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 03 0 003 001 0025 TD2993D0          0046 D0 D0 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 03 0 003 002 0025 TD3093D1          0047 D1 D1 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 03 0 003 003 0025 TD310391          0048 91 91 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 03 0 003 004 0025 TD3113AK          0049 AK AK TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 03 0 003 005 0025 TD312912          0050 12 12 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 04 0 004 000 0026 TD316316          0051 16 16 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 04 0 004 001 0026 TD3163AA          0052 AA AA TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 04 0 004 002 0026 TD3203M7          0053 M7 M7 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 05 0 005 000 0027 TD325385          0054 85 85 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 05 0 005 001 0027 TD3263A7          0055 A7 A7 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 05 0 005 002 0027 TD3273C5          0056 C5 C5 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 06 0 006 000 0028 TD3853QP          0057 QP QP TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 06 0 006 001 0028 TD390303          0058 03 03 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 06 0 006 002 0028 TD405367          0059 67 67 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 06 0 006 003 0028 TD406387          0060 87 87 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 06 0 006 004 0028 TD407370          0061 70 70 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 07 0 007 000 0029 TD449346          0062 46 46 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 07 0 007 001 0029 TD449347          0063 47 47 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 07 0 007 002 0029 TD453348          0064 48 48 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 07 0 007 003 0029 TD453349          0065 49 49 TD830TERM           ASYN 9600 TDCLASS           SYSTEM/CANDE
109 114 08 0 008 000 0001 RJESYSB771        0002 00 00 RJE9600SYSTERM      ASYN 9600 TDCLASS           SYSTEM/RJEII
109 114 08 0 008 001 0001 RJEB771SC1        0003 01 01 RJE9600SPOTERM      ASYN 9600 TDCLASS           SYSTEM/RJEII
109 114 08 0 008 002 0001 RJEB771CR1        0004 02 02 RJE9600READERTERM   ASYN 9600 TDCLASS           SYSTEM/RJEII
109 114 08 0 008 003 0001 RJEB771LP1        0005 03 03 RJE9600PRINTTERM    ASYN 9600 TDCLASS           SYSTEM/RJEII
```

The following example shows a portion of what would be produced on the terminal of a B 5900 when the GRAPH option is specified. A chart of the datacomm network is given showing the NSP, LSPs, designated lines, and line stations. The chart includes the type of terminal associated with each line station along with the logical station number (LSN) for each terminal. Other information such as line adaptor numbers is also given.

```
         SYSTEM/DCSTATUS (3.3.321)  DATE : 02/03/83  TIME :  1408:39

          FILES: B59/010483/NIF ON DISK, B59/010483/DCPCODE ON DISK

          DATE OF NDL COMPILATION : 01/04/83 , TIME : 0000:00

          NO RECONFIGURATION EXECUTED.

***********     ************      *************    ***************
*         *     *          *      * DL = 0:1  *    * DLS = 0:1:0 *
* NSP 108 ******* LSP 112  ******** TDLINE    ****** TD19920HG   *
*         *  *  *          *   *  * ADAPTER=0 *  * * TD830TERM   *
*         *  *  *          *   *  *           *  * * LSN = 4     *
***********  *  ************   *  *************  * ***************
             *                 *                 *
             *                 *                 * ***************
             *                 *                 * * DLS = 0:1:1 *
             *                 *                 *** TD19920G1   *
             *                 *                 * * TD830TERM   *
             *                 *                 * * LSN = 5     *
             *                 *                 * ***************
             *                 *                 *
             *                 *                 * ***************
             *                 *                 * * DLS = 0:1:2 *
             *                 *                 *** TD19920HE   *
             *                 *                 * * TD830TERM   *
             *                 *                 * * LSN = 6     *
             *                 *                 * ***************
             *                 *                 *
             *                 *                 * ***************
             *                 *                 * * DLS = 0:1:3 *
             *                 *                 *** TD19920HF   *
             *                 *                 * * TD830TERM   *
             *                 *                 * * LSN = 7     *
             *                 *                 * ***************
             *                 *                 *
             *                 *                 *
```

The following example shows the output from the DCSTATUS option TERMINAL on the line printer or at a remote terminal connected to a B 5900. The NDL settings for the terminals are given in categories such as MAXINPUT, MAXOUTPUT, RECEIVE-ADDRESS-SIZE, and so forth.

```
        SYSTEM/DCSTATUS (3.4.140)   DATE : 05/18/83   TIME : 1003:49
                  FILES: MV69/D/NIF ON DISK, MV69/D/DCPCODE ON DISK
                  DATE OF NDL COMPILATION : 05/05/83 , TIME : 0000:00
                  NO RECONFIGURATION EXECUTED.

 TERMINAL    = TD830TERM
     UNIQUE-ID                   = 1
     MAX-INPUT                   = 1920
     MAX-OUTPUT                  = 1920
     RECEIVE-ADDR-SIZE           = 2
     TRANSMIT-ADDR-SIZE          = 2
     RECEIVE-DELAY               = 0
     TRANSMIT-DELAY              = 0
     TERMINAL-TYPE               = TD830TYPE
     PAGE-SIZE                   = 24
     PAGE-COUNT                  = 2
     LINE-WIDTH                  = 80
     SCREEN                      = TRUE
     WRAP-AROUND                 = TRUE
     CLASS                       = TDCLASS
          UNIQUE-ID                   = 6
          MODE                        = ASYNC.
          VERTICAL-PARITY             = EVEN
          HORIZONTAL-PARITY           = TRUE
               CRC-POLYNOMIAL         = X7+1
               CRC-INITIAL            = 0
               CRC-FINAL              = 0
          BIT-RATE                    = 9600
          STOPBITS-TYPE               = LONG
```

The following example shows the output from the DCSTATUS option STATION on a B 5900 line printer. The station name and terminal name are given along with information about the DCC station table.

```
        SYSTEM/DCSTATUS (3.4.140)   DATE : 05/18/83   TIME : 1003:49
                  FILES: MV69/D/NIF ON DISK, MV69/D/DCPCODE ON DISK
                  DATE OF NDL COMPILATION : 05/05/83 , TIME : 0000:00
                  NO RECONFIGURATION EXECUTED.

 STATION 3

 STATION NAME = ADMSTA
 TERMINAL NAME = ADMTERM
 DCC STATION TABLE
 500040030050   WRAPAROUND : ENABLED : READY :
 MCS = 1: LSN = 3 : WIDTH = 80
 6F000A64003E   STATION REMOTE TYPE = 0: RETRY COUNT = 10:NIF INDEX = 62
 000000000000
 PRIMARY QUEUE = 0,CURRENT Q = 0,STN Q = 0,PSEUDOMCS=0
 000000000000   DLS : UNASSIGNED
```

5 DUMPANALYZER

5.1 GENERAL INFORMATION

The SYSTEM/DUMPANALYZER utility produces user-specified subsets of information from a memory dump and analyzes that information according to parameters given by default or supplied by the user.

This section is intended as a reference source for experienced programmers who are familiar with the workings of the Master Control Program (MCP). Because no single memory dump is typical, no attempt is made here to explain how a memory dump is read. This ability can be acquired only through experience and a thorough knowledge of the system software.

System types are identified within this section as follows:

HDU	-	B 7900, A 15
IOM	-	B 7700, B 7800
MLIP	-	A 3, A 9, A 10, B 5900, B 6900

DUMPANALYZER FILES

SYSTEM/DUMPANALYZER uses three files: OPTIONS, TAPEIN, and MCPCODEFILE. File-equating these files is sometimes desirable when DUMPANALYZER is run using a Command AND Edit (CANDE) language or Work Flow Language (WFL) RUN statement or using Menu-Assisted Resource Control (MARC).

OPTIONS

OPTIONS is the file name of the user input file. When DUMPANALYZER is run from an ODT, OPTIONS should be file-equated to SPO. For example:

```
RUN SYSTEM/DUMPANALYZER;FILE OPTIONS(KIND=SPO)
```

TAPEIN

The memory image created by the memory dump routine in the Master Control Program (MCP) is stored in a file named MEMORY/DUMP.

When DUMPANALYZER is run it accepts as input a file titled TAPEIN. That file can be file-equated to any of the following kinds of files.

a. A file that is titled MEMORY/DUMP, created when the system takes a dump.

b. A "pseudorecovery" file that has the title

```
DP/<MMDDYY>/<HHMM>/<REASON_ID>
```

where MMDDYY represents month, day, and year and HHMM represents hour and minutes.

c. A SAVEd dump file, created using the DUMPANALYZER SAVE command.

MCPCODEFILE

The code file of the MCP that was running at the time of the dump. This file contains the MCP names and index arrays (and may contain LINEINFO information) for analyzing stack bases, program information blocks (PIBs), and file information blocks (FIBs). The file may be successfully file-equated only to MCP code files closely related to the code file that took the dump. (The code file must differ by only a few patches.) This file is not required when a previously SAVEd file is equated to TAPEIN.

PSEUDORECOVERY FILE

During the initialization sequence of DUMPANALYZER, the MEMORY/DUMP file is accepted as input and certain data structures are built up. When initialization is complete, this data structure information is saved in a "pseudorecovery" file under the user's usercode. This file has the name

```
DP/<MMDDYY>/<HHMM>/<REASON_ID>
```

where MMDDYY represents month, day, and year, and HHMM represents hour and minutes.

If the DUMPANALYZER run is interrupted by a Halt/Load or terminated by an operator, this pseudorecovery file can be file-equated to TAPEIN and used as input to DUMPANALYZER.

If for any reason the user wishes to exit from the DUMPANALYZER session, considerable time and resource savings will be made if the RECESS command is used rather than the STOP or BYE command. The RECESS command does not remove the pseudorecovery file, while the other two commands do. In all cases DUMPANALYZER does not remove the file that was used as input.

The user should also be aware that the pseudorecovery file is removed if the SAVE command is issued and successfully completed. To save creation time and disk space, the SAVE command should be used only if the MCP code file information will not be available at a later time. In all other cases, the user should use the pseudorecovery file as input and DUMPANALYZER will pick up the relevant MCP information while going through initialization.

If SYSTEM/DUMPANALYZER is running with FAMILY substitution in effect, the program first looks for MCP and program code files without FAMILY specifications; it then looks for such files with FAMILY specifications.

See also

"SAVED" Memory Dumps. 50
SAVE. 161

"SAVED" MEMORY DUMPS

It is usually advisable to take a raw memory dump and run the SAVE command on it in a separate DUMPANALYZER session before the main analysis of the dump is performed. Running SAVE on the dump provides more complete MCP information for the eventual analysis than would be available if DUMPANALYZER were run on a raw dump. When a SAVEd dump is analyzed, the MCP running on the system need not be identical to the MCP running when the dump was taken. No file equation of the MCP code file is necessary when running DUMPANALYZER on a SAVEd dump.

In the DUMPANALYZER session when the SAVE command is run, the MCP that is file-equated to MCPCODEFILE should be identical to the one that was running on the system when the raw dump was taken. After the SAVE command is run on the raw dump, the resultant disk file is called a SAVEd dump. The SAVEd dump contains the memory image from the dump tape and the relevant data from the MCP code file. The SAVEd dump is usually an extremely large file and is often stored on tape.

When the DUMPANALYZER session for the main analysis is initiated, the SAVEd dump file is file-equated as the TAPEIN file, and the MCPCODEFILE does not need to be file-equated.

MCP LEVEL COMPATIBILITY

SYSTEM/DUMPANALYZER checks to determine the difference between the DUMPANALYZER Mark level and the MCP Mark level on the dump tape. If the levels are not the same, DUMPANALYZER terminates with an error message similar to the following:

```
    CANNOT ANALYZE nn MEMORY DUMP WITH mm DUMPANALYZER
```

where nn is the MCP Mark level and mm is the DUMPANALYZER Mark level.

In some cases, SYSTEM/DUMPANALYZER allows analysis of a wrong level MCP when the run-time option MEMONLY is chosen. The error message indicates when this is the case. It is always better to use the correct level DUMPANALYZER.

If the MCP code file does not have the same timestamp as the MCP at the time of the memory dump, the following error message is displayed:

```
    ACCEPT: WRONG CODE FILE--OK OR RESTART
```

The operator enters OK if DUMPANALYZER should continue using the same MCP code file. If RESTART is entered, the code file is closed and SYSTEM/DUMPANALYZER will look for a file on DISK titled MCPCODEFILE. The operator should then use the FA (File Attribute) Operator Display Terminal (ODT) command to specify the desired MCP code file.

5.2 RUNNING DUMPANALYZER

DUMPANALYZER can be run from a remote terminal, ODT, or card reader. Each command is processed before the following command has been parsed.

REMOTE OPERATION

To initialize a DUMPANALYZER remote operation, the following command is entered from a CANDE or MARC session:

```
    RUN *SYSTEM/DUMPANALYZER
```

A MARC menu selection for running DUMPANALYZER also exists.

When necessary, file-equate the input dump file to TAPEIN and the MCP that was running to MCPCODEFILE in the RUN statement.

After the RUN statement, DUMPANALYZER displays the following messages:

```
    DUMPANALYZER VERSION 36.000.00000
    SELECT RUN TIME OPTIONS: PRINTER, REMOTE, DEBUG, MEMONLY
```

Enter the desired option and transmit. To select the default option, transmit a blank line. DUMPANALYZER then initializes. When DUMPANALYZER is ready to accept commands it displays the prompt ":READY".

Semantics:

By default, the output is routed to the remote terminal. PRINTER causes output to be routed to the printer. REMOTE causes output to be routed to the remote terminal. DEBUG causes diagnostic information related to DUMPANALYZER to be displayed. MEMONLY allows a restricted analysis of memory dumps.

Examples:

```
    RUN *SYSTEM/DUMPANALYZER ON MYPACK;
          FILE TAPEIN(KIND=DISK,TITLE=ER/DUMP)

    R *SYSTEM/DUMPANALYZER;VALUE=1;FILE TAPEIN(SERIALNO="10046");FILE
        MCPCODEFILE(KIND=DISK,TITLE=SYSTEM/CMP34180)
```

ODT OPERATION

Three methods can be used to initiate DUMPANALYZER using the ODT. Only limited output is sent to the ODT, such as the HELP command information. Output from most commands is sent to the printer as soon as the session terminates. If the RELX command is in effect, information is sent to the printer during the session.

Method 1

In the first method, enter the following:

```
RUN SYSTEM/DUMPANALYZER; FILE OPTIONS(KIND=SPO)
```

When necessary, file-equate the input dump file to TAPEIN and the MCP that was running to MCPCODEFILE in the RUN statement.

The following message is then displayed:

```
INITIALIZING
---------------------------------------------------
FUNCTIONS CURRENTLY AVAILABLE ARE
```

A list of commands is then displayed, and the system displays the following message:

```
"HELP" FOR THIS LIST, "HELP HELP" FOR MORE INFO
```

DUMPANALYZER continues to initialize and then displays the following message:

```
ENTER REQUESTS
```

DUMPANALYZER commands can now be entered.

Method 2

The second method uses the DA (Dump Analyzer) ODT command. Refer to the Operator Display Terminal (ODT) Reference Manual for a complete description of the DA command.

Enter the appropriate form of the DA command and transmit. The following message is then displayed:

```
INITIALIZING
---------------------------------------------------
FUNCTIONS CURRENTLY AVAILABLE ARE
```

A list of commands is then displayed, and the system displays the following message:

```
"HELP" FOR THIS LIST, "HELP HELP" FOR MORE INFO
```

DUMPANALYZER continues to initialize and then displays the following message:

```
ENTER REQUESTS
```

DUMPANALYZER commands can now be entered.

Method 3

If DUMPANALYZER is run on an ODT configured in datacomm mode, it should be initiated using MARC. This is done by entering "??MARC", logging on to MARC, and proceeding with the REMOTE OPERATION instructions.

CARD READER

The following execution deck is entered at the card reader:

```
?RUN SYSTEM/DUMPANALYZER
?DATA OPTIONS
      .
      .
% COMMAND LIST
      .
      .
?END
```

All output is printed on the line printer.

Example:

```
BEGIN JOB ANALYSIS;
RUN SYSTEM/DUMPANALYZER;
DATA OPTIONS
SUMMARY
LOCKS
DEADLOCK
IO UINFO NAMES ALL
OPT
MEM
TRACE
BOXINFO
SWAPANAL
MODE + ALL
ALLSTACKS SUMMARY ACTIVE
NAMES
NETWORK
AREAS AVAIL CODE LINKS DESC
DC
?
END JOB
```

5.3 INPUT TO DUMPANALYZER

The following subsection, "Basic Constructs", describes the syntactic variables commonly used in the addressing and value schemes specified in various DUMPANALYZER commands. "DUMPANALYZER Commands" in this section provides detailed descriptions of each of the commands that can be used as input to DUMPANALYZER.

5.3.1 BASIC CONSTRUCTS

The addressing and value schemes used in the syntax diagrams of DUMPANALYZER commands commonly employ certain basic syntax constructs. These basic constructs are defined as follows:

<number>

```
----<hexadecimal number>-------|
  |                            |
  |- DEC --<decimal number>-|
  |                            |
  |- OCT --<octal number>---|
```

Semantics:

<hexadecimal number>

A number in base 16, each of whose digits ranges from 0 to F. In DUMPANALYZER commands, <number> is assumed to be hexadecimal unless it is preceded by a prefix such as DEC (decimal) or OCT (octal), indicating another base.

<decimal number>

A number in base 10, each of whose digits ranges from 0 to 9. In DUMPANALYZER commands, <number> is decimal when it is preceded by the prefix DEC.

<octal number>

A number in base 8, each of whose digits ranges from 0 to 7. In DUMPANALYZER commands, <number> is octal when it is preceded by OCT.

SIMPLE ADDRESS

A <simple address> represents a location in memory. A <simple address> is made up of an <absolute address> or a <simple location> followed by an optional offset. Three types of <simple location>s exist: stack-related, global, and indirect. <offset> is a number that indicates the displacement of the <simple address> from the given <simple location> or <absolute address>.

<simple address>

```
    ----<absolute address>------------------------|
      |                    | |                |
      |-<simple location>--| |- + ---<offset>-|
                             |     |
                             |- - -|
```

<absolute address>

```
    --<number>--|
```

<simple location>

```
    ----<stack id>-----------------------------------|
      |             |                             |
      |             |-<stack offset>--------------|
      |             |                             |
      |             |- BASE ---<attribute name>---|
      |                      |                    |
      |                      |- # --<offset>------|
      |                                           |
      |- PIB --<stack number>---<attribute name>--|
      |                        |                  |
      |                        |- # --<offset>----|
      |                                           |
      |- G ---<global id>-------------------------|
      |     |                                     |
      |     |- # --<offset>-----------------------|
      |                                           |
      |- RV --<simple value>----------------------|
      |                                           |
      |- VIA --<ASD number>-----------------------|
```

<stack id>

```
---- STK --<hexadecimal stack number>-------------|
  |                                 | |       |
  |- @ ------------------------------| |- SD -|
```

<stack offset>

```
---- LOSR ----|
  |       |
  |- BOSR -|
  |       |
  |- SREG -|
```

<offset>

```
--<number>--|
```

Semantics:

<absolute address>

A hexadecimal, decimal, or octal number that specifies an address within a present, on-line memory module. The prefixes DEC and OCT are required if decimal or octal numbers are specified. The address specified by <hexadecimal address>, <decimal address>, and <octal address> must be in the range from 0 to 4"FFFFF".

<simple location>

Specifies locations in memory that are stack-relative, global identifiers, or indirect.

The following groups of tokens represent valid <simple location>s; expansions of some of the tokens within those groups are included:

<stack id>

A <simple location> may consist of a <stack id>. A <stack id> specifies a stack by number or uses the last stack explicitly referenced. An expansion of the token <stack id> follows:

```
STK <hexadecimal stack number>
STK <hexadecimal stack number> SD
@
@ SD
```

These four statements are all expansions of the token <stack id>. STK indicates that a stack is involved. <hexadecimal stack number> specifies the hexadecimal number that identifies the stack. The last stack explicitly referenced is represented by the "at" sign (@). Referencing a stack by its hexadecimal number sets up the @ for subsequent use. The segment dictionary of the stack may be referenced if the stack number or the "at" sign (@) is followed by SD.

```
<stack id> <stack offset>
```

A <simple location> can consist of a <stack id> (as defined previously), followed by a specific location within the stack, which is called the <stack offset>. An expansion of the token <stack offset> follows:

LOSR	The Limit of Stack Register.
BOSR	The Bottom of Stack Register.
SREG	The S Register.

```
<stack id> BASE <attribute name>
<stack id> BASE <offset>
PIB <stack number> <attribute name>
PIB <stack number> <offset>
```

For some other possible <simple location>s, the stack BASE and program information block (PIB) of the stack may also be used; a <stack id> (with BASE) or <stack number> (with PIB) must be specified. In addition, either an <offset> or an attribute name must be identified. The <attribute name>s are listed in the MCP symbolic; they change with each new release. Since the availability of the cell names depends on the presence of the MCP code file, these cell names are not valid if the task (PIB) and stack index arrays cannot be generated.

G <global id>
G <offset>

These are <simple location>s that are global ids (MCP D[0] cells). The G denotes that the location is global. <offset> indicates a displacement of a given number of words away from D[0].

RV <simple value>

Addresses can also be indirectly specified by using the RV ("the reference value"), option. For example, if a cell contains an absolute address, an indirect reference word (IRW), or a stuffed indirect reference word (SIRW), the RV option allows the contents of this cell to be used as an indirect address. (An IRW would require that an environment be set up, but an SIRW would not; a set up environment implies that a stack has already been referenced.) An absolute address would be used as is. The actual value that can be used for indirect addressing is a <simple value>, which is defined later in this subsection. Each level of indirection is printed as it occurs so that each chain of addresses can be seen along with the referenced data.

VIA <ASD number>

A <simple location> pointed to by an <ASD number>. <ASD number> can be any number from 1 through the largest valid ASD that is specified in ASD1[0].

+ <offset>
- <offset>

An <offset> is a hexadecimal, decimal, or octal number that specifies the number of words away from a reference point such as a <simple location>, that a particular address is located. If an <offset> is specified in decimal form, DEC should precede it. If an <offset> is specified in octal form, OCT should precede it. The offset either increases (+) or decreases (-) the address.

Examples:

In the following examples, blank spaces and special characters function as delimiters. Delimiters are needed whenever two alphanumeric items are juxtaposed.

```
41AC0                       %ABSOLUTE ADDRESS

DEC 47990                   %ABSOLUTE ADDRESS

OCT 170071                  %ABSOLUTE ADDRESS

STK 4A SD BASE BDINFO       %BASE ADDRESS (STACK-RELATED)

PIB 2E7 SERIAL              %PIB ADDRESS (STACK-RELATED)

PIB 2E7 # 5F                %PIB ADDRESS (STACK-RELATED)

@ SD LOSR                   %INVARIANT STACK-RELATED ADDRESS

G HLUNIT                    %GLOBAL IDENTIFIER ADDRESS

RV M[47AB]                  %INDIRECT ADDRESS
```

MULTIPLE ADDRESSES

<multiple addresses> provides for the specification of more than one address. The addresses generated by these expressions are treated as sequences; that is, the first address in the sequence is generated and passed to the execution routine that requires it. Then, the next address is generated, and so forth, until the list is exhausted. The addresses are generated in the proper order, with memory addresses increasing, except in the case of stack-related addresses, in which case, the addresses are generated from LOSR to BOSR. These rules hold true even when the specification by the user differs from the proper order.

<multiple addresses>

```
      |<---------------- , ---------------|
      |                                   |
    ----<simple address>---------------------|
                         |              |
                         |-<until part>-|
```

<until part>

```
    ---- FOR ---<simple value>----|
      |       |                |
      |       |- ALL ----------|
      |                        |
      |- TO ---<offset>--------|
             |                 |
             |-<stack offset>--|
             |                 |
             |- END -----------|
```

Semantics:

<until part>

The <until part> specifies over what range a list of <multiple addresses> should extend, from an initial location.

FOR ALL

Indicates that all the addresses from the initial location to the end of the area referenced by the descriptor that provided the initial location are to be included as part of the <multiple addresses>.

FOR <simple value>

Here the <simple value> indicates the number of consecutive addresses to be included in the <multiple addresses> list. <simple value> is defined later in this subsection.

TO <offset>

Here the <offset> indicates the displacement of the highest address to be included in that part of the <multiple addresses>.

TO <stack offset>

Here the <stack offset> indicates that all addresses up to the indicated location in the stack (BOSR, LOSR, SREG) are to be included as part of the <multiple addresses>.

TO END

Indicates that all the addresses from the initial location to the end of the area referenced by the descriptor that provided the initial location are to be included as part of the <multiple addresses>.

Pragmatics:

Only one <absolute address> per statement is allowed. Multiple stack references must reference the same stack.

Examples:

The following are examples of possible <multiple addresses>. Each example uses a different form of the <until part>.

```
3BAC FOR 20

3BAC TO 3BCB

STK 32 LOSR TO BOSR

RV STK 11C BASE SWAPHOLD FOR ALL
```

SIMPLE VALUE

A <simple value> is a single scalar value either defined by the user or derived from a value in a memory location or in a stack. A <simple value> is a <word value> followed by an optional <concatenation> value.

```
<simple value>

    --<word value>-------------------------------|
                   |                          |
                   | |<---------------------| |
                   | |                      | |
                   |--- & --<concatenation>---|


<word value>

    --<simple word>--------------------------|
                    |                     |
                    |- . --<partial word>-|


<simple word>

    ---- M -- [ --<hexadecimal address>-- ] ------------------|
      |                                                    |
      |- C -- ( ---<simple location>---- ) ----------------|
      |          |                     |                   |
      |          |-<absolute address>-|                    |
      |                                                    |
      |-<number>-------------------------------------------|
      |                                                    |
      |-<simple location>---- [ --<simple index list>-- ] -|
      |                     |                              |
      |-<absolute address>-|                               |
      |                                                    |
      |- ( --<simple value>-- ) ---------------------------|


<simple index list>

      |<------ , ------|
      |                |
    ----<simple value>----|
```

<partial word>

```
    ---- [ --<simple value>-- : --<simple value>-- ] ----|
      |                                               |
      |- TAG -----------------------------------------|
```

<concatenation>

```
--<word value>----------------------------------------------------->

>--- [ --<left bit to>-- : --<left bit from>-- : --<bit count>-- ] ----|
  |                                                                |
  |- [ --<left bit to>-- : --<bit count>-- ] ----------------------|
  |                                                                |
  |- TAG ----------------------------------------------------------|
```

<left bit to>

```
    --<simple value>--|
```

<left bit from>

```
    --<simple value>--|
```

<bit count>

```
    --<simple value>--|
```

Semantics:

<word value>

A <word value> is a <simple word> with an optional <partial word>. The following groups of tokens represent valid <simple word>s; expansions of these tokens are also given.

M [<simple address>]

"M" signifies memory. The contents of the memory location at <hexadecimal address> is one type of <simple word>. This operation is the same as subscripting the MEMORY array.

C (<simple location>)
C (<absolute address>)

"C" signifies contents. The C option may be used to obtain the contents of any <simple location> or <absolute address>. <simple location> and <absolute address> are defined under <simple address>, one of the preceding "Basic Constructs".

<number>

A <simple word> may be specified as a hexadecimal, decimal, or octal number.

<simple location> [<simple index list>]
<absolute address> [<simple index list>]

A <simple location> or <absolute address> may be indexed, and the derived value may be used as a <simple word>. This method of forming values is valid only if the word specified (which can be reached through an IRW chain) is an unindexed data descriptor, and the number of indexes specified matches the number of dimensions in the array.

<partial word>

A <partial word> is an optional component of a <word value>. When a partial word is present, the <word value> is the value of a selected group of bits within the <simple word>. <partial word> specifies a particular group of bits within the <simple word>.

<simple value>:<simple value>

Two <simple value>s separated by a colon (:) make up a <partial word>. The first <simple value> indicates the number of the starting bit in a range. The second <simple value> indicates how many bits the range extends over (heading from 47 down to 0).

TAG

When TAG is specified as the <partial word>, then <word value> is the value of the 3-bit TAG in the attached <simple word>.

<concatenation>

<simple value> consists of a <word value> & <concatenation>. In the context of concatenation, <word value> is a 48-bit, binary word. <concatenation> includes within itself a second <word value>, along with bit specifiers and a bit count; the specifiers and count indicate a substitution of certain bits to be made from the second word value into the first. For further information about bit manipulation see the ALGOL Reference Manual chapter on "Expressions" under "<concatenation>".

<word value> [<left bit to>:<left bit from>:<bit count>]

Here, <word value> represents a binary, 48-bit word with a three-bit tag value. The word value in the <concatenation> is the "source word", while the word value that preceded the ampersand (&) is the "destination word". <left bit to> defines the highest (ranging from 0-47) bit number in the destination word. <left bit from> defines the highest (ranging from 0-47) bit location in the source word. The <bit count>, ranging from 1-48, specifies the length of the data field to be moved from the source word to the destination word.

<word value> [<left bit to>:<bit count>]

<word value> represents a binary, 48-bit word with a 3-bit tag value. <left bit to> defines the highest (ranging from 0-47) bit number in the destination word. In this case, the <left bit from> specification is understood to coincide with the <left bit to> specification. Again, <bit count> specifies the length of the data field to be moved from the source word to the destination word.

<word value> TAG

Here <word value> represents a binary, 48-bit word with a 3-bit tag value. TAG indicates that the TAG of the source word should be substituted for the TAG in the destination word.

Examples:

In the following examples, blank spaces and special characters function as delimiters. Delimiters are needed whenever two alphanumeric items are juxtaposed.

```
M[47AC]                %MEMORY LOCATION

C (G HLUNIT)           %CONTENTS OF SIMPLE LOCATION

DEC 123456 & 3 TAG     %CONCATENATION

C (STK 53 BOSR).[6:2]  %PARTIAL WORD
```

5.3.2 DUMPANALYZER COMMANDS

The following describes the commands that can be used when running SYSTEM/DUMPANALYZER. These commands allow the user to select the type of analyses to be done on the dumped data. Multiple commands separated by semicolons can be entered on the same line.

ACBTABLEBASE

The ACBTABLEBASE command displays the base of the ACB table or specifies a new <simple address> for future computations. An error message is displayed if this command is used on non-MCP/AS systems.

<acbtablebase>

```
    -- ACBTABLEBASE -----------------------|
                      |                  |
                      |-<simple address>-|
```

Semantics:

ACBTABLEBASE

This form of the command displays the base of the ACB table.

ACBTABLEBASE <simple address>

This form of the command sets the base of the ACBTABLE to the specified <simple address> for future computations.

ALLSTACKS

The ALLSTACKS command causes an analysis of stacks in the system to be printed. If the MODE command is previously selected, the 8-bit arrays are displayed in EBCDIC format as well as hexadecimal format.

<allstacks>

```
    -- ALLSTACKS ------------------------------------|
                 |                                  |
                 |           |<-----------------|  |
                 |           |                  |  |
                 |- SUMMARY ----------------------|
                               |              |
                               |- ACTIVE ----|
                               |              |
                               |- DMSIIJOBS -|
                               |              |
                               |- DUMPING ---|
```

Semantics:

ALLSTACKS

All stacks in the system are analyzed.

ALLSTACKS SUMMARY ACTIVE

A full analysis of all stacks that were alive at the time of the dump is given, with summary information given for all other stacks.

ALLSTACKS SUMMARY DMSIIJOBS

A full analysis of task stacks that were using DMSII, the database stack, and the database task stack is given, with summary information given for all other stacks.

ALLSTACKS SUMMARY DUMPING

A full analysis of the stack that took the dump is given, with summary information given for all other stacks.

AREAS

The AREAS command prints the contents of memory areas. Areas are continuous portions of memory surrounded by clusters of words that serve as memory links.

<areas>

```
    -- AREAS -------------------------------------------|
               |                                      |
               | |<--------------------------------|  |
               | |                                 |  |
               |----- AVAIL --------------------------|
                   |                             |
                   |- CODE ----------------------|
                   |                             |
                   |- DESC ----------------------|
                   |                             |
                   |- DOUBLE --------------------|
                   |                             |
                   |- LINKSONLY -----------------|
                   |                             |
                   |- ODDBALL --<oddballfield>---|
                   |                             |
                   |- RANGE --<multiple address>-|
                   |                             |
                   |- SIZE --<number>------------|
                   |                             |
                   |- STATS ---------------------|
                   |                             |
                   |- STATSONLY -----------------|
```

<oddballfield>

```
---- BUFFERHEADER -----|
  |                  |
  |- DIRECTIOBUFF --|
  |                  |
  |- DOPEVECT ------|
  |                  |
  |- DSKHDR --------|
  |                  |
  |- EVENTARRAY ----|
  |                  |
  |- FIBMARK -------|
  |                  |
  |- IOCBAREA ------|
  |                  |
  |- NORMALAREA ----|
  |                  |
  |- OLDSIBMARK ----|
  |                  |
  |- PIBMARK -------|
  |                  |
  |- SEGDOPE -------|
  |                  |
  |- SEGSEG --------|
  |                  |
  |- SIBMARK -------|
  |                  |
  |- SORTAB --------|
  |                  |
  |- STACKMARK -----|
  |                  |
  |- SWAPSPACEMARK -|
```

Semantics:

AREAS

The contents of all memory areas that are not available, code, or read-only are given.

AREAS AVAIL

The contents of available memory areas are given.

AREAS CODE

The contents of resident code segments and read-only data areas are given.

AREAS DESC

A descriptor analysis is performed on all areas that are printed.

AREAS DOUBLE

The output from the AREAS command is double-spaced.

AREAS LINKSONLY

LINKSONLY inhibits the printing of the contents of the memory areas, and only the links around the memory areas are given.

ODDBALL refers to one of the fields in the memory link words that gives information about the contents of the memory area. The command "AREAS ODDBALL <oddballfield>" prints the contents of all memory areas that contain the items specified in the <oddballfield>.

The items that can be found in the <oddballfield> are defined as follows:

BUFFERHEADER	The area contains information about input/output (I/O) buffers.
DIRECTIOBUFF	The area contains a direct array.
DOPEVECT	The area contains a dope vector; this dope vector includes mom descriptors.
DSKHEADER	The area contains information about disk file headers.
EVENTARRAY	The area contains an event array.
FIBMARK	The area contains a file information block (FIB).
IOCBAREA	The area contains an Input/Output Control Block (IOCB).
NORMALAREA	The area is a "normal" area that does not contain any of the structures indicated in the rest of the option list.
OLDSIBMARK	The area contains a Data Management System II (DMSII) Structure Information Block (SIB) created on Mark 3.4 or earlier releases.

PIBMARK	The area contains a program information block (PIB) or task variable, or a segment dictionary pseudo-PIB.
SEGDOPE	The area contains the dope vector for a segmented (that is, "paged") array.
SEGSEG	The area is a page of a segmented array.
SIBMARK	The area contains a DMSII "SIB" created on Mark 3.5 or later releases.
SORTAB	The area contains a sort table listing absolute addresses.
STACKMARK	The area contains a stack or a segment dictionary.
SWAPSPACEMARK	The area contains space suballocated by the SWAPPER utility.

AREAS RANGE <multiple address>

The analysis of memory areas is restricted to the specified address range. Areas partially or completely in the desired range are analyzed. The omission of the RANGE modifier causes all areas in all memory subsystems to be analyzed.

AREAS SIZE <number>

The contents of all memory areas consisting of <number> words are printed.

AREAS STATS

Statistics regarding the memory areas associated with each stack are collected. These statistics are reported in a memory usage summary.

When using the STATS option repeatedly, totals are reset and not accumulated.

AREAS STATSONLY

All area displays are omitted and only the final statistics are given.

ARRAYLIMIT

The ARRAYLIMIT command limits the size of each array printed to the number of lines specified by the positive <decimal number>.

<arraylimit>

```
    -- ARRAYLIMIT --<decimal number>--|
```

Semantics:

If the <decimal number> is 0, the printing of arrays is suppressed.

For multidimensional arrays, the limit is applied cumulatively for both the dope vector entries and the data entries.

ASDNUMBER

The ASDNUMBER command prints out information about ASDs. An error message is displayed if this command is used on a non-MCP/AS system.

<asdnumber>

```
    -- ASDNUMBER ---<number>-------------------------------------|
                  |          |                   | |            |
                  |          |- UNTIL <number> --| |- EXPAND -|
                  |          |                   |
                  |          |- FOR <number> ----|
                  |                              |
                  |- STACK --<number>------------|
                                      |          |
                                      |- VIRGIN -|
```

Semantics:

ASDNUMBER <number>

This form of the command prints out ASD1 through ASD4 for the specified ASD number.

ASDNUMBER <number> UNTIL <number>

This form of the command prints out ASD1 through ASD4 for the specified range of ASDs.

ASDNUMBER <number> FOR <number>

This form of the command prints out ASD1 through ASD4 for the number of ASDs specified starting at the designated ASD number.

ASDNUMBER STACK <number>

This form of the command prints a list of the ASDs for the specified stack.

ASDNUMBER STACK <number> VIRGIN

This form of the command prints a list of the virgin ASDs associated with the specified stack.

EXPAND

When the EXPAND parameter is used, the information is printed in greater detail.

ASDTABLEBASE

The ASDTABLEBASE command displays the base of the ASD table or specifies a new <simple address> for future computations. An error message is displayed if this command is used on non-MCP/AS systems.

<asdtablebase>

```
    -- ASDTABLEBASE -----------------------|
                   |                   |
                   |-<simple address>-|
```

Semantics:

ASDTABLEBASE

This form of the command displays the base of the ASD table.

ASDTABLEBASE <simple address>

This form of the command sets the base of the ASDTABLE to the specified <simple address> for future computations.

<u>ASN</u>

The function of the ASN (Addressing Space Number) command is equivalent to that of the BOX command. ASN, like "box", refers to a memory subsystem in an extended memory system (such as the B 7900) or a tightly-coupled system. The ASN command specifies all local addresses referred to in any following commands as residing in the ASN specified by <number>. ASN may be substituted for BOX on all systems. BOX is an invalid command on B 7900 and A 15 systems; thus, ASN is the only choice for those systems.

This command is invalid on systems using MCP/AS.

<asn>

```
    -- ASN -----------------------|
            |                   |
            |-<decimal number>-|
```

Semantics:

ASN

The current setting for the ASN is displayed.

ASN <decimal number>

All local addresses are interpreted as residing in the ASN specified by <decimal number>. The <decimal number>s constructs are specified either in the configuration file or by the EC Operator Display Terminal (ODT) command (refer to the Operator Display Terminal (ODT) Reference Manual for further information).

Pragmatics:

A STACK command overrides an ASN setting. A stack command for a stack in a local box causes ASN to be set to that local box. (Refer to "Stack".)

See also

BOX (IOM SYSTEMS)	80
STACK	168

BOX (IOM SYSTEMS)

Each memory subsystem in a tightly-coupled system is called a "box". The BOX command indicates that all addresses referred to in any following commands are in the box specified by <number>. The BOX command can also be used to interrogate the current box setting. BOX is a synonym for ASN. The BOX command cannot be used on A 15 and B 7900 systems; the ASN command must be used instead. This command is invalid on systems using MCP/AS.

<box>

```
    -- BOX ---------------|
            |           |
            |-<number>-|
```

Semantics:

BOX

Use of the word BOX alone, without a number, displays the current setting; the response is displayed on the ODT when running DUMPANALYZER from the ODT.

BOX <number>

BOX followed by a number sets the box number. <number> designates which box is being referred to. The "Global Memory Subsystem" is box 0. The box number of a local memory subsystem is its associated processor id number.

Pragmatics:

A STACK command for a stack in a local box automatically changes the box setting to the number of the box the stack is in.

BOXINFO

The BOXINFO command causes the box information arrays to be printed for each box in the system. Each array is printed as a raw array.

```
<boxinfo>

    -- BOXINFO --|
```

Example:

The following is a partial example of the box information arrays returned by the BOXINFO command:

```
DUMP OF BOXINFO[3]
FC79A/00000  0 000000 000000    ...
FC7A1/00007  0 000000 000000    ...
FC7AB/0000E  0 000000 000000    ...
FC7AF/00015  0 000000 0039A2    ...
FC7B6/0001C  0 000000 000000    ...
FC7BD/00023  0 000000 000000    ...
FC7C4/0002A  2 000000 080528    ...
FC7CB/00031  2 0000CC 6A7869    ...
FC7D2/00038  0 000000 04CA1A    ...
FC7D9/0003F  0 000066 2B77E2    ...
```

CAND

The CAND command analyzes a "candidate" for a port-matching structure. The candidate is a subport that has been offered for interprocess communication and has not yet found a matching subport. (Refer to the example under "PORT".) The CAND command is only valid for BNA Version 1.

```
<cand>

    -- CAND ---<index>------------|
             |                  |
             |- AT --<address>-|
```

Semantics:

CAND <index>

The subport with the indicated <index> is analyzed. The index is found in the library info word of the File Information Block (FIB).

CAND AT <address>

The subport at the given <address> is analyzed.

See also

PORT. 143

CODEINFO (HDU SYSTEMS)

The CODEINFO command displays the attributes of code memory for the code environment of the current ASN.

```
<codeinfo>

    -- CODEINFO --|
```

Semantics:

If the current ASN is not split, this command has no effect. The CODEINFO command displays the current ASN's code memory management table, which is divided into memory areas. The information displayed for each area appears as follows:

```
   <code addr> <table entry> <mom/link word> <attribute word>
               <attributes for this code area>
```

Attributes of a code area include available, in-use, saved, currently saved, to be saved, orphaned, and guardian.

DC

The DC command causes a full data communications analysis to be printed. (Refer to Figure 5-1.)

<dc>

```
   -- DC --------------------|
         |                |
         |- DCP ---------|
                 |         |
                 |- MSG -|
```

Semantics:

The DCP option causes an analysis of the DCP tables to be printed. This analysis includes DCP line vectors, DCP line tables, and DCP station tables in addition to the DCC table analysis.

The MSG option causes messages in nontanked datacomm queues in the DCALGOL queue to be analyzed.

```
INPUT: DC

                                                   DATA COMMUNICATIONS ANALYSIS
DATACOM CONFIGURATION:
    MAXIMUM LSN       = 356
    DCPS CONFIGURED  = 2, 3
    DCPS INITIALIZED = 2, 3
    DCPS 2 AND 3 ARE EXCHANGED
    DCP[2]: HEYU CABLE ON MPX #2, DCP #1; LINES=0; LOCAL MEMORY TABLES; TEST Q = 000; DCC SNR=049
    DCP[3]: HEYU CABLE ON MPX #8, DCP #1; LINES=0; LOCAL MEMORY TABLES; TEST Q = 000; DCC SNR=048
MCS INFORMATION:
    MCS[1]: SYSTEM/CANDE/PACK
          STACK=082, PRIMARY QUEUE=000E
    MCS[2]: SYSTEM/RJE
          NOT RUNNING, NO PRIMARY QUEUE
    MCS[3]: SYSTEM/DIAGNOSTICMCS/SYS32
          DIAGNOSTICMCS, NOT RUNNING, NO PRIMARY QUEUE
            .
            .
            .
    MCS[13]: SYSTEM/RJEII
          STACK=20D, PRIMARY QUEUE=002D

DCC STATION TABLE
LSN 2: ----- NO LINE ASSIGNMENT -----
          0 100040 020048  (ENABLED, READY, NOT/ATTACHED, MCS=1, WIDTH=72)
          0 6F030F 0001C3  (CONTROLCHAR=4"6F", REMOTETYPE=3, RETRY=15, APPLICATION INDEX=0, NIF INDEX=451)
          0 000000 000000  (QUEUES: PRIMARY=NONE; CURRENT=NONE; FILE=NONE; PSEUDOMCS=0)
          0 000000 000000  (DLS=NONE)
            .
            .
            .
LSN 5: DLS=3:90:0
          0 140040 050048  (ENABLED, READY, ATTACHED, MCS=1, WIDTH=72)
          0 6F040F 0001C6  (CONTROLCHAR=4"6F", REMOTETYPE=4, RETRY=15, APPLICATION INDEX=0, NIF INDEX=454)
          0 000000 00E00E  (QUEUES: PRIMARY=000E; CURRENT=000E; FILE=NONE; PSEUDOMCS=0)
          0 000000 835A00  (DLS=3:90:0)
LSN 6: DLS=3:92:0
          0 140040 060048  (ENABLED, READY, ATTACHED, MCS=1, WIDTH=72)
          0 6F040F 0001C7  (CONTROLCHAR=4"6F", REMOTETYPE=4, RETRY=15, APPLICATION INDEX=0, NIF INDEX=455)
          0 000000 00E00E  (QUEUES: PRIMARY=000E; CURRENT=000E; FILE=NONE; PSEUDOMCS=0)
          0 000000 835C00  (DLS=3:92:0)

DCC LINE TABLE FOR DCPS 2 AND 3
LINE 0: (CLUSTER ADDRESS=0:0, DESCRIPTOR ADDRESS=8DE92)
          5 C00000 28DF28  LINE DESCRIPTOR
          0 800001 01031C  (READY, MAXSTAS=1, REALSTAS=1, NIF INDEX=796)
          5 C00000 48E0D5  STATION 0: LSN=15
LINE 1: (CLUSTER ADDRESS=0:1, DESCRIPTOR ADDRESS=8DE93)
          5 C00000 28DF2D  LINE DESCRIPTOR
          0 800501 01031E  (READY, MAXSTAS=1, REALSTAS=1, NIF INDEX=798)
                           SWITCHED: NOT/CONNECTED, AUTOANSWER)
          5 C00000 48E0F1  STATION 0: LSN=22
```

Figure 5-1. Data Communications Analysis

<u>DCP</u>

The DCP command is similar to the DC command except that it analyzes only the DCP tables, not the DCC tables.

```
<dcp>

    -- DCP ------------|
            |       |
            |- MSG -|
```

Semantics:

The MSG option causes messages in nontanked datacomm queues in the DCALGOL queue to be analyzed.

See also
    DC. 84

<u>DCTRACE</u>

The DCTRACE command displays or prints a summary of the last 250 requests or results of the specified Data Communication Processor (DCP)/Network Support Processor (NSP). These requests or results are listed in the chronological order they were processed by the DCCONTROL stack of the specified DCP/NSP; thus, requests and results are interspersed in the summary. The summary includes such items as request type, associated logical station number (LSN) or DCP number, line number, station number (DLS) and associated error values.

```
<dctrace>

    -- DCTRACE --<relative DCP/NSP number>--|
```

Examples:

The following are examples of DCTRACE input and the subsequent output on an NSP system and a DCP system.

NSP System Example

```
    Input: DCTRACE 3

    Output:

    <timestamp> REQUEST 7: ADD EDITOR REQUEST FOR EDITOR 1
                           LENGTH 3944

    <timestamp> RESULT  7: ACK RESULT IN RESPONSE TO REQUEST NUMBER 7
                           LENGTH 10

    <timestamp> REQUEST 8: OUTPUT REQUEST FOR LSN 285, TEXT LENGTH 32
                           LENGTH 68

    <timestamp> REQUEST 9: OUTPUT REQUEST FOR LSN 286, TEXT LENGTH 32
                           LENGTH 68

    <timestamp> RESULT  8: OUTPUT STATUS RESULT FOR LSN 285 IN
                                        RESPONSE TO REQUEST NUMBER 8
                           LENGTH 28

    <timestamp> RESULT  9: OUTPUT STATUS RESULT FOR LSN 286 IN
                                        RESPONSE TO REQUEST NUMBER 9
                           LENGTH 28
```

DCP System Example

Input: DCTRACE 2

Output:

```
<timestamp> REQUEST:  SET STATION TYPE(23) FOR DLS 2:0D:00
                      LENGTH 48

<timestamp> RESULT:   FOR SET STATION TYPE(23) REQUEST FOR DLS
                                                       2:0D:00
                      RESULT BYTE INDEX IS GOOD RESULT

<timestamp> REQUEST:  WRITE REQUEST(7) FOR DLS 2:0D:00, TEXT
                                                     LENGTH 22
                      LENGTH 60

<timestamp> REQUEST:  WRITE REQUEST(7) FOR DLS 2:0D:01, TEXT
                                                     LENGTH 22
                      LENGTH 60

<timestamp> RESULT:   FOR WRITE REQUEST(7) REQUEST FOR DLS
                                                       2:0D:00
                      RESULT BYTE INDEX IS GOOD RESULT

<timestamp> RESULT:   FOR WRITE REQUEST(7) REQUEST FOR DLS
                                                       2:0D:00
                      RESULT BYTE INDEX IS GOOD RESULT
```

DEADLOCK

The DEADLOCK command causes information to be printed regarding stacks that hold locks or are waiting for a lock. The following items are analyzed.

a. Global locks

b. Header locks

c. Directory locks

d. Userdisk locks

e. I/O waits (on MPX systems only)

f. Operator Reply Waits

If a stack is waiting on any of these items, the stack is printed. In addition, any of the locks that the stack holds are listed.

All EVENTs and EVENT ARRAYs declared in the MCP outer block are considered, but "hard" locks are not reported.

A stack is reported as waiting for a unit or path only when the stack is actually waiting, and not when the stack simply has I/Os requested.

<deadlock>

```
                 |<--------------------|
                 |                     |
-- DEADLOCK ---/1\--------------------|
                     |             |
                     |- CRITICAL -|
                     |             |
                     |-<number>---|
```

Semantics:

DEADLOCK

All stacks that either hold locks or are waiting for a lock are listed by stack number, along with the names of the events or locks.

DEADLOCK CRITICAL

CRITICAL is specified to indicate that only stacks that hold locks and are also waiting on locks are to be listed. CRITICAL deadlocks may indicate an MCP problem.

DEADLOCK <number>

<number> specifies a particular stack. When a <number> is given, only locks for the stack identified by <number> are listed.

Example:

The following is a sample of the information that is printed regarding stacks that either hold locks or are waiting for locks:

```
INPUT: DEADLOCK

LOCK AND WAITING STACK ANALYSIS

STACK 01E WAITING FOR EVENT 262 ANABOLEVENT
STACK 048 WAITING FOR EVENT 3CB DCPHEYU[0003]
STACK 049 WAITING FOR EVENT 3CB DCPHEYU[0002]
STACK 04B WAITING FOR IO VIA UNIT 00E (14)
STACK 28D WAITING FOR REPLY #2
STACK 363 WAITING FOR IO VIA UNIT 0C4 (196)
          PROCURED EVENT 355 HEADERLOCKS[0066]
```

DEBUG

The DEBUG command dynamically sets, disables, or interrogates the DEBUG option. If DEBUG is set, a program dump is produced to aid in debugging SYSTEM/DUMPANALYZER.

```
<debug>

    -- DEBUG ----------|
              |     |
              |- + -|
              |     |
              |- - -|
              |     |
              |- ? -|
```

Semantics:

DEBUG
DEBUG +

Sets the DEBUG option.

DEBUG -

Disables the DEBUG option.

DEBUG ?

Shows whether or not the DEBUG option is set.

Example:

The following DEBUG command sets the DEBUG option:

```
   DEBUG +

         DEBUG SET
```

FIB

The FIB command causes DUMPANALYZER:

a. to assume that a file information block (FIB) starts at an arbitrary address, and

b. to analyze that FIB.

The FIB specified at <simple address> is printed. Text contained in buffers is printed unconditionally.

<fib>

```
    -- FIB ----------<simple address>--|
           |      |
           |- AT -|
```

FIB <simple address>
FIB AT <simple address>

The FIB starting at <simple address> and an analysis of that FIB are printed.

GC (MLIP SYSTEMS)

The GC command displays the configuration of the system at the time of the dump. The resources of the system and information about the I/O subsystem are given, including all DLPs base by base for all bases and DLPs configured at the time of the dump. The GC command is valid only for MLIP systems.

```
<gc>

    -- GC --|
```

Example:

The following is an example of the GC command display and an explanation of some of its terminology.

INPUT: GC

```
     ***** GROUP CONFIGURATION *****
GROUP ID: SYSTEMA
PROCESSORS:
          PROC ID. 2;
OPERATIONS:
          DL BACKUP   DISK;
          DL LOG      DISK;
          DL USERDATA DISK;
          DL JOBS     DISK;
          DL CATALOG  DISK;
          DL OVERLAY  DISK;
          DL SORT     DISK;
          HN          SYSTEMA;
PERIPHERALS ALLOWED TO GROUP:
   5,29-33,61,64,71,108,112,117,132-134,140;
I/O:
   BASE 0/2/0
      HOST 2     % PATH = MLIP PORT 0  LEMPORT 0
         ADDRESS 1 DLPID 8   CR1    %%%% DLP STATUS = ABSENT
         ADDRESS 2 DLPID 132 ODT1
         ADDRESS 3 DLPID 5   TP1
         ADDRESS 4 DLPID 60  HT1
         ADDRESS 6 DLPID 140 HC2
         ADDRESS 7 DLPID 28  MT1;
   BASE 1/2/0
      HOST 2     % PATH = MLIP PORT 2  LEMPORT 0
         ADDRESS 4 DLPID 108 NSP3;
   BASE 2/3/0
      HOST 2     % PATH = MLIP PORT 3  LEMPORT 0
         ;       % NO DLPS ASSIGNED TO THIS HOST
   BASE 3/2/0
      HOST 2     % PATH = MLIP PORT 1  LEMPORT 0
         ADDRESS 1 DLPID 112 LSP1;
      DEPENDENT HOST 4 DLPID 108      %% DC STATUS = NOT SEEN
         ADDRESS 1 DLPID 112 LSP1
         ADDRESS 2 DLPID 117 LSP3;     %%%% DLP STATUS = NOT SEEN
   BASE 3/3/0
      DEPENDENT HOST 6 DLPID 111      %% DC STATUS = NOT SEEN
         ADDRESS 0 DLPID 115 LSP1     %%%% DLP STATUS = NOT SEEN
         ADDRESS 1 DLPID 113 LSP1     %%%% DLP STATUS = NOT SEEN
         ADDRESS 2 DLPID 124 LSP1     %%%% DLP STATUS = NOT SEEN
         ADDRESS 4 DLPID 121 LSP1     %%%% DLP STATUS = NOT SEEN
         ADDRESS 5 DLPID 122 LSP1     %%%% DLP STATUS = NOT SEEN
         ADDRESS 6 DLPID 123 LSP1;    %%%% DLP STATUS = NOT SEEN
```

PROC ID.
A unique number identifying a host processor

PATH
A description of the unique path from a host to a unit's DLP, consisting of an MLIP and a LEM port

MLIP PORT
The relative Message-Level Interface Port (MLIP) that the path traverses (there can be up to eight ports per processor)

LEMPORT
The relative Line Expansion Module (LEM) port that the path traverses

BASE
A description of the base in which the unit's DLP resides

DEPENDENT HOST
If a path to a unit or units is through an outboard host (such as an NSP), a description of the outboard host

GRAPHS

The GRAPHS command causes MCP stack graphs to be printed one level deep.

<graphs>

```
    -- GRAPHS ---------------|
               |           |
               |-<number>-|
```

Semantics:

GRAPHS

All MCP stack graphs are printed one level deep.

GRAPHS <number>

The graphs for the stack identified by <number> are printed.

HARDINFO (HDU SYSTEMS)

The HARDINFO (Hardware Information) command displays various system state registers and data. HARDINFO is valid only when analyzing a standalone tape dump for an extended memory system.

<hardinfo>

```
    -- HARDINFO --|
```

HDR

The HDR command causes an analysis of the disk file header stack or a particular header and prints out that analysis. (Refer to Figure 5-2.)

```
<hdr>

    -- HDR ----------------------------|
            |                         |
            |- SUMMARY -------------|
            |                         |
            |- ROWS ----------------|
            |                         |
            |- < number > ----------|
                           |          |
                           |- ROWS -|
                           |          |
                           |- RAW --|
```

Semantics:

HDR

The HDR form of the command prints an analysis of all the file headers that existed in memory when the dump was taken. Row address words are not printed.

HDR SUMMARY

The HDR SUMMARY form of the command prints a list containing the location and names of all the headers that existed in memory when the dump was taken.

HDR ROWS

The HDR ROWS form of the command produces a printout with all the information that the HDR form produces plus an analysis of the row address words.

HDR <number>

The HDR <number> form of the command prints the same information as HDR, but only for the header specified by <number>.

HDR <number> ROWS

The HDR <number> ROWS form of the command prints the same information as HDR ROWS, but only for the header specified by <number>.

HDR <number> RAW

The HDR <number> RAW form of the command produces a printout in hex format of the entire header specified by <number>.

Pragmatics:

The first time that DUMPANALYZER encounters the HDR command without the RAW, ROWS, or SUMMARY option the following message is printed:

```
*** WARNING *** : ROW ADDRESS WORDS PRINTED ONLY WITH THE ROWS OPTION
```

This warning will be removed in the Mark 3.7 Release.

```
INPUT: HDR 4F
HDR [004F]  AT 5 800003 77843E  (LENGTH = 55) NAME : (SITE)TESTUSR1 ON DISK
HDR0[004F]  =  0 000900 000000   LENGTH = 0, LOCATION = 0, BASE UNIT = 9
         0000  0 3F3F06 E00000  BLOCKLENGTH = 55, LOCATION = 0
         0001  0 0010C0 612000  OPENCOUNT = 1, FILEKIND = 192(DATA), WRITTEN ON, FIXEDSIZE = 18
         0002  0 048000 0C0000  FIBINFO (EXTMODE = EBCDIC, FILETYPE = 0, SIZE: MODE = 0/OFFSET = 0/SIZE = 0)
         0003  0 005000 000050  DISKBLOCKING (BLOCKSIZE = 80, MINRECSIZE = 0, MAXRECSIZE = 80)
         0004  0 3AAA0F AE12B8  TIMESTAMP (DATE = 85018, TIME = 2:48:21.7376)
         0005  0 630014 0003E8  VERSION = 6, SECURITY = 3, R/W SECURITY = 0, MODE = 0, ROWS = 20, ROWSIZE = 1000
         0006  0 000001 000000  SAVEFACTOR = 0, AVAILSPACE = 0, FILEORG = 0
         0007  0 080200 000000  OPT ATTRIBUTES = 2, TIMESTAMPSYNCF, FLATHDRLENGTH = 0
         0008  0 011033 009001  TITLEINDEX = 51, TITLESIZE = 17, BASEUNIT = 9, LASTUNIT = 1
         0009  0 000000 000000  DISK EOFU = 0, DISK EOFV = 0
         000A  0 3AAA0F AE0CB9  CREATION TIMESTAMP (DATE = 85018, TIME = 2:48:21.6787)
         000B  0 3AAA0F AE0CB9  ALTER TIMESTAMP (DATE = 85018, TIME = 2:48:21.6787)
         000C  0 3AAA0F AE0CB9  ACCESS TIMESTAMP (DATE = 85018, TIME = 2:48:21.6787)
         000D  0 0000C0 000000  MULTIUSE WORD
         000E  0 000000 8B004F  ORIGHDRVERSION = 0, CONTENDORS = 0, OPENER = 139, COREINDEX = 79
         000F  0 0000C0 000000  NOT USED
         0010  0 0000C0 000000  NOT USED
         0011  0 0000C0 000000  NOT USED

         NUMBER OF ALLOCATED ROWS = 1

         OPTIONAL ATTRIBUTE WORDS:

         0000  0 080200 420028  ATTNO = 2 (SECURITY GUARD), TYPE = BYTE LIST, SIZE = 66, OFFSET = 40
                                VALUE : (JAGAN)TESTING/OPTIONAL/ATTRIBUTES/SECURITY/GUARD/AND/USER/INFO.

         0001  0 000300 000400  ATTNO = 3 (USERINFO), TYPE = FIELD
                                VALUE : 00000400 (DEC) : 1024
```

Figure 5-2. Disk File Header Analysis

<u>**HEADING**</u>

The HEADING command causes the heading (first page) of a dump to be printed at the terminal or line printer.

The information provided in the header includes the title of the MCP, the cause of the memory dump, and the processor that initiated the dump.

```
<heading>

    -- HEADING --|
```

Example:

The following is an example of the information returned by the HEADING command:

```
  INPUT: HEADING

       DUMPANALYZER VERSION: 34.542.177
       ****************************************
       *          B 6800 MEMORY DUMP          *
       *       (SYSTEM SERIAL: # 123)         *
       *     B6900 MCP: MARK 34.542.3022      *
       *      5/15/83          23:08          *
       * THIS IS THE ANALYZED DUMP FROM MT 15 *
       *          TAPE SERIAL: DMP857         *
       ****************************************
       CLOCK = 0000010CC80C (04:27:32 SINCE HALT/LOAD)
                         ACTUAL CLOCK = 000815AAC121 (23:08:55)
       DUMPING MCP:  *SYSTEM/MCP ON DISK.
                     CREATION TIMESTAMP: 09/15/83 22:12:34
       ANALYZED MCP: *SYSTEM/MCP ON SYS228
                     CREATION TIMESTAMP: 09/16/83 21:04:37


       HOSTNAME:  XYZ3
       INTRINSIC NAME:   (NOT SPECIFIED)
       CAUSE OF DUMP:   CONTROLLER DIED
       MEMDUMP CALLED @ 0DA4:0026:4 (42844740) DCATTACHDETACH
       MEMORY DUMP PERFORMED BY:    PROCESSOR 4 IN STACK 00B  S=0B13B
```

DUMPANALYZER

ITEM	DESCRIPTION
A.	Cause of dump: the value following "MEMDUMP CALLED @" specifies the segment and relative address where memdump was called.
B.	Memory dump performed by: the processor that initiated the dump, the stack number in hexadecimal, and the S register setting. Any other active processors are then printed with their stack numbers and S register settings.

When the memory dump procedure is invoked, it "seizes" all processors with a processor-to-processor (HEYU) interrupt. This information indicates the position of the processors at the time they were seized.

HELP

The HELP command provides information about DUMPANALYZER commands. When DUMPANALYZER is run from the ODT, the response to HELP is displayed on the ODT.

```
<help>

    -- HELP -----------------------------|
             |                          |
             |-<dumpanalyzer command>-|
             |                          |
             |-<meta-item>------------|
```

Semantics:

HELP

A list of available DUMPANALYZER commands is given.

HELP <dumpanalyzer command>

A railroad syntax diagram and a brief explanation of the commands is given.

HELP <meta-item>

A further explanation of most metalinguistic items (for example, <number>) is provided. The broken brackets are required for meta-items.

IO

The IO command invokes input/output (I/O) analysis of all peripherals. (Refer to Figures 5-3 and 5-4 for examples of output from the IO command.)

<io>

```
    -- IO ----------------------------------------------------------->

    >---------------------------------------------------------------|
       |                                 | |           | |           |
       ||<------------------------------| | |- SUMMARY -| |- RESULTQ -|
       ||                               | |                          |
       |-----/1\-<unit spec.>------------|                           |
       |   |                         |                               |
       |   |-/1\- UINFO -------------|                               |
       |                 |         |                                 |
       |                 |- NAMES -|                                 |
       |                 |         |                                 |
       |                 |- ALL ---|                                 |
       |                                                             |
       |-<active spec.>----------------------------------------------|
```

<unit spec.>

```
    -- UNIT ----------------<decimal number>----|
             |            |                  |
             |- INTERNAL -|                  |
             |            |                  |
             |- LOGICAL --|                  |
             |                               |
             |-<unit type>-------------------|
             |                               |
             |- CATALOG ---------------------|
             |                               |
             |- OFFLINEUNIT -----------------|
             |                               |
             |- VOLUNIT ---------------------|
             |                               |
             |- TAPEUNIT --------------------|
```

<active spec.>

```
    -- ACTIVE ---------------------------------------------|
                |                                          |
                |- UNIT ----------------<decimal number>-|
                         |             |
                         |- INTERNAL -|
                         |             |
                         |- LOGICAL --|
```

<unit type>

```
    ---- MT ----|
      |       |
      |- PK -|
      |       |
      |- DK -|
      |       |
      |- HC -|
      |       |
      |- DC -|
      |       |
      |- SC -|
      |       |
      |- LP -|
      |       |
      |- CP -|
      |       |
      |- CR -|
      |       |
      |- DT -|
      |       |
      |- EN -|
      |       |
      |- IP -|
      |       |
      |- HY -|
      |       |
      |- SP -|
```

AX (Accept) Message Command of the IO command

```
-- ? -- AX --- WHERE ----|
           |         |
           |- STOP --|
           |         |
           |- SKIP --|
```

Semantics:

IO

Each I/O peripheral is analyzed and result queues are displayed.

IO ACTIVE

On non-MLIP systems, any unit with a non-empty I/O queue is considered "active." It can also be assigned to an MCP stack.

On MLIP systems (B 5900, B 6900, A 3, A 9, and A 10), the IO ACTIVE form searches for active IOCBs; this search can be asynchronously monitored and controlled by entering the AX (Accept) message command after the IO ACTIVE form has been initiated.

The WHERE option displays the location of the search, the number of the physical unit now being searched, and the number of active IOCBs found so far.

The STOP option cancels the search and prompts the user with the ":READY" message; no information from the search is saved.

The SKIP option stops the search of the current unit and skips to the next unit (if any). The search occurs from the high end of memory toward 0; thus, memory indices printed in response to ?AX WHERE decrease.

The IO ACTIVE form of the command has no effect when analyzing a dump created by a B 7900 or A 15 system.

IO UNIT <decimal number>

The IO UNIT <decimal number> form of the command causes the analysis of the designated I/O peripheral. Each I/O peripheral has a unique physical unit number associated with it. It is that number that is used as the decimal number in this form of the command.

The IO UNIT form of the command can also indicate whether the unit number used is an internal unit number or a logical unit number. If neither INTERNAL nor LOGICAL is used, the <decimal number> is assumed to be an external unit number. An example of this form of the command is "IO UNIT INTERNAL 27".

IO UNIT <unit type>

This form of the command causes the analysis of the designated I/O peripheral unit type.

IO UNIT CATALOG
IO UNIT OFFLINEUNIT
IO UNIT VOLUNIT
IO UNIT TAPEUNIT

These forms of the command provide specific table information concerning catalog units. To give significant information, these forms of the command must be followed by a UINFO option.

IO UINFO
IO UINFO NAMES
IO UINFO ALL

If the UINFO option is specified by itself, a unit information analysis is provided. A basic analysis of each I/O peripheral, identical to that provided by the IO form of the IO command, is given; in addition, a UINFO entry for each peripheral is given. The UINFO option lists the contents of each UINFO entry as an uninterpreted array. When the UINFO NAMES option is used, it provides a columnar listing of the entries and the purpose of each word. The mass-storage lists and other arrays attached to the UINFO entry are not printed. These lists can be obtained by using the UINFO ALL option. The UINFO ALL option applies only to pack and disk.

IO RESULTQ

The IO RESULTQ form of the command provides the user with result queues that are otherwise suppressed. This option is only valid on MLIP and HDU systems. An example of this option is "IO UNIT MT RESULTQ".

IO SUMMARY

The IO SUMMARY form of the command provides a skeleton analysis of the specified I/O peripheral unit(s). The unit table and I/O queue expansion are suppressed from the output. An example of the use of this option is "IO UNIT MT RESULTQ SUMMARY".

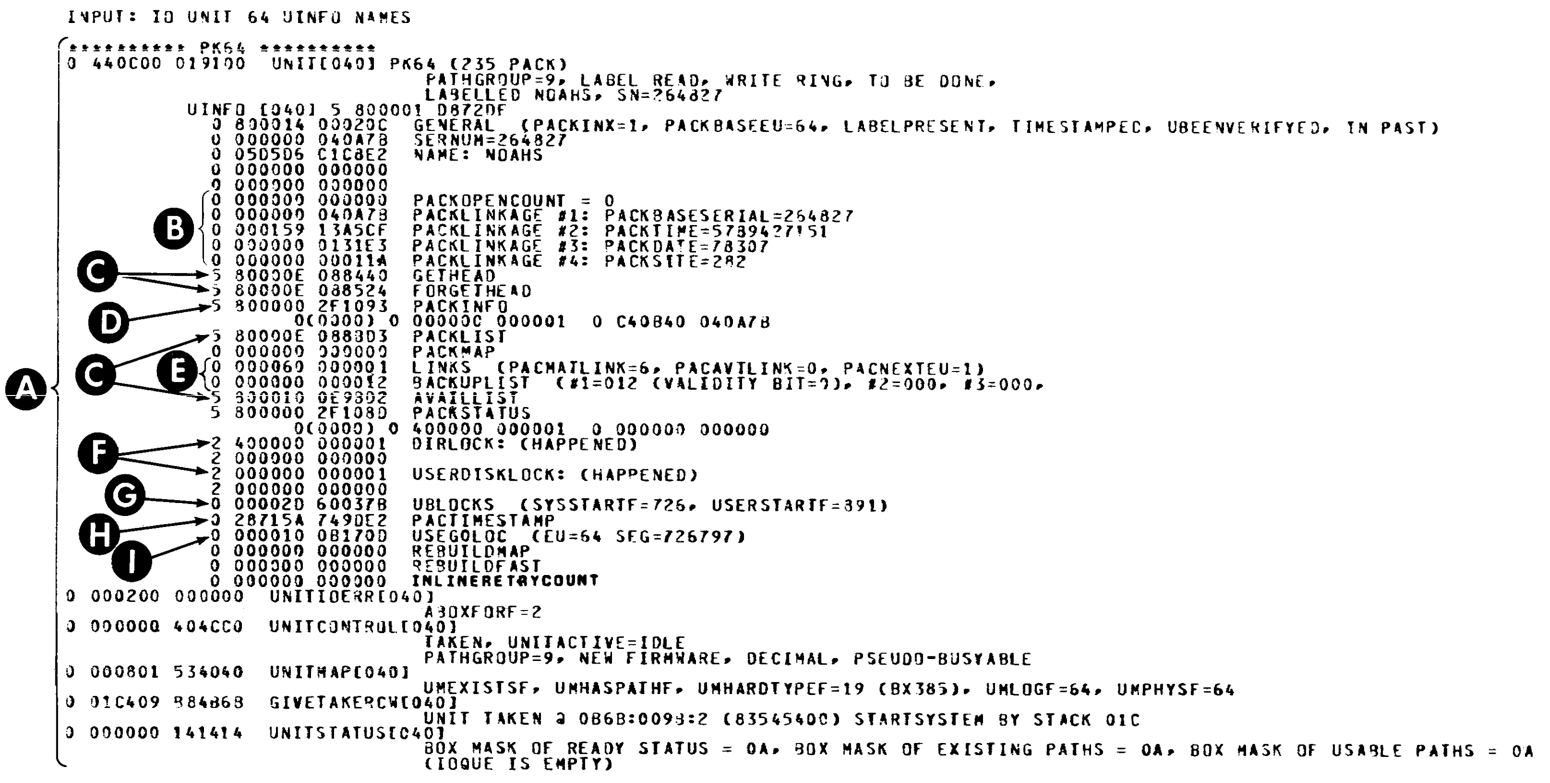
```
INPUT: IO UNIT 58 UINFO NAMES

(UNIT @ FE81F, UNITMAP @ FE60F, GIVETAKERCW @ FD3AA, UNITCONTROL @ FE717, } A
 UINFO @ FE233, UNITIOERR @ FE498, IOQUE @ FD932, PATHQUE @ FD743,
 UNITSTATUS @ FE353)

********** SC58 **********
0 030000 033000  UNIT[03A] SC58 (TD830 SPO)
                          PATHGROUP=7, FILE ASSIGNED (STACK=053),
                          LABELLED
           UINFO [03A] 5 800001 D880F3
                 0(0000) 0 AB0002 400000  0 000001 000001  0 000000 000000  0 0208C0 000000  0 000000 000000  0 000000 000700
                 6(0006) 0 000000 000000  0 000000 0000C0  0 010000 000000  0 000C00 001000  0 000000 000000  0 000000 000000
     B          12(000C) 0 000000 000008  0 000000 000000  0 000000 000000  0 008508 240000  0 000000 C00000  0 000000 000000
                18(0012) 0 000000 000000  THRU 20(0014)
             C  21(0015) 0 000000 000019  0 000000 00001B  0 070101 03D9C5  0 D40000 000000  0 070101 03D9C5  0 D40000 000000
                27(0018) 0 040102 020000  0 021100 000000
0 053400 000000  UNITIOERR[03A]
                          STKNRF=053, ABOXFORF=4
0 000000 433800  UNITCONTROL[03A]
                          TAKEN, UNITACTIVE=I/O BUSY
                          PATHGROUP=7
0 000801 423A3A  UNITMAP[03A]
                          UMEXISTSF, UMHASPATHF, UMHARDTYPEF=2, UMLOGF=58, UMPHYSF=58
0 000000 101010  UNITSTATUS[03A]
                          BOX MASK OF READY STATUS = 08, BOX MASK OF EXISTING PATHS = 08, BOX MASK OF USABLE PATHS = 08
                        { 0 008814 688146  IOQUE[03A] (1ST IOCB=88146, LAST IOCB=88146)
                          .... IOCB @ 88146
                          0 10CB0A 885300  IOMISC  (MARK=10CB, CHAR MODE, IOCW IN BUFFER, INITIATE, THRU IOREQUEST, INITIATOR S
                          0 006FE0 530000  IOUNITS  (I/O LENGTH = 1790 EBCDIC CHARACTERS, BILLING STACK = 053)
                          0 000003 000000  IOMASK  (MASKMPXNUMF=3)
                          0 000000 000000  PHYSICALRD
                          0 004012 A8A5AF  AREADESC  (ADDRESS=8A5AF, LENGTH=1790 EBCDIC CHARACTERS)
                          M[8A5AF]: 0 030000 000E82  (IOCW: IOSTANDARDFIELD: WRITE, 8-BIT, MEMORY PROTECT)
                          M[8A5B0]: 0 3C0000 000000  (WORD  0 OF BUFFER)
                          M[8A5B1]: 0 000000 000000  (WORD  1 OF BUFFER)
                      D { M[8A5B2]: 0 000000 000000  (WORD  2 OF BUFFER)
                          0 40018E A98989  TIMECELL  (INITIATED 6568111.2 MICROSECONDS PRIOR TO DUMP)
                          5 E10000 28A5A5  EVNT
                          5 C00003 C032BA  FIBDESC
                          0 100000 000040  IOCBCONTROL  (SUBSYSTEMF=1, CANDOITF=4)
                          0 000000 000000  RDDETAIL
                          0 000000 000000  RDEXTENDED
                          0 030000 000E82  IOCW
                          0 000000 000000  LOGICALRD
                          5 C4007D 08A5B0  TEMPLATE
                          0 05303A 000018  IOSTUFF  (OWNER STACK=053, UNIT=58, IOINPROCESSF, USERIO)
                        { 0 000000 000000  IOLINKAGE
```

ITEM	DESCRIPTION
A	STARTING ABSOLUTE MEMORY ADDRESSES OF PERIPHERAL UNIT-RELATED INFORMATION. THIS INFORMATION INCLUDES THE UNIT TABLE, UNITMAP TABLE, GIVETAKERCW TABLE, UNITCONTROL TABLE, UNIFO TABLE, UNITIOERR TABLE, IOQUE, PATHQUE, AND UNITSTATUS TABLE.
B	CONTENTS OF THE WORD INDEXED IN THE UNIT TABLE.
C	CONTENTS OF THE DESCRIPTOR INDEXED IN THE UINFO TABLE.
D	ABSOLUTE MEMORY ADDRESS OF THE IOCB. THE FORMAT OF THE IOCB SHOWN IS FROM A B 6800; THE IOCB FORMAT FOR THE B 6900 IS DIFFERENT.

Figure 5-3. Peripheral Unit Analysis (Beginning)



```
INPUT: IO UNIT 64 UINFO NAMES

********** PK64 **********
0 440C00 019100   UNIT[040] PK64 (235 PACK)
                              PATHGROUP=9, LABEL READ, WRITE RING, TO BE DONE,
                              LABELLED NOAHS, SN=264827
          UINFO [040] 5 800001 D872DF
            0 800014 00020C  GENERAL  (PACKINX=1, PACKBASEEU=64, LABELPRESENT, TIMESTAMPED, UBEENVERIFYED, IN PAST)
            0 000000 040A7B  SERNUM=264827
            0 05D5D6 C1C8E2  NAME: NOAHS
            0 000000 000000
            0 000000 000000
            0 000000 000000  PACKOPENCOUNT = 0
            0 000000 040A7B  PACKLINKAGE #1: PACKBASESERIAL=264827
            0 000159 13A5CF  PACKLINKAGE #2: PACKTIME=5789427!51
            0 000000 0131E3  PACKLINKAGE #3: PACKDATE=78307
            0 000000 00011A  PACKLINKAGE #4: PACKSITE=282
            5 80000E 088440  GETHEAD
            5 80000E 088524  FORGETHEAD
            5 800000 2F1093  PACKINFO
                  0(0000) 0 00000C 000001  0 C40B40 040A7B
            5 80000E 0883D3  PACKLIST
            0 000000 000000  PACKMAP
            0 000060 000001  LINKS  (PACMAILINK=6, PACAVILINK=0, PACNEXTEU=1)
            0 000000 000012  BACKUPLIST  (#1=012 (VALIDITY BIT=0)), #2=000, #3=000,
            5 800010 0E9302  AVAILLIST
            5 800000 2F108D  PACKSTATUS
                  0(0000) 0 400000 000001  0 000000 000000
            2 400000 000001  DIRLOCK: (HAPPENED)
            2 000000 000000
            2 000000 000001  USERDISKLOCK: (HAPPENED)
            2 000000 000000
            0 00002D 60037B  UBLOCKS  (SYSSTARTF=726, USERSTARTF=391)
            0 28715A 749DE2  PACTIMESTAMP
            0 000010 0B170D  USEGOLOC  (EU=64 SEG=726797)
            0 000000 000000  REBUILDMAP
            0 000000 000000  REBUILDFAST
            0 000000 000000  INLINERETRYCOUNT
0 000200 000000   UNITIOERR[040]
                              ABOXFORF=2
0 000000 404CC0   UNITCONTROL[040]
                              TAKEN, UNITACTIVE=IDLE
                              PATHGROUP=9, NEW FIRMWARE, DECIMAL, PSEUDO-BUSYABLE
0 000801 534040   UNITMAP[040]
                              UMEXISTSF, UMHASPATHF, UMHARDTYPEF=19 (BX385), UMLOGF=64, UMPHYSF=64
0 01C409 884868   GIVETAKERCW[040]
                              UNIT TAKEN @ 086B:0098:2 (83545400) STARTSYSTEM BY STACK 01C
0 000000 141414   UNITSTATUS[040]
                              BOX MASK OF READY STATUS = 0A, BOX MASK OF EXISTING PATHS = 0A, BOX MASK OF USABLE PATHS = 0A
                              (IOQUE IS EMPTY)
```

ITEM	DESCRIPTION
A	CONTENTS OF THE WORD INDEXED IN THE FOLLOWING TABLES: UNIT, UNITIOERR, UNITCONTROL, UNITMAP, GIVETAKERCW, UNITSTATUS.
B	PACKOPENCOUNT: INCREMENTED WHEN A REQUEST IS MADE AND DECREMENTED WHEN A REQUEST HAS BEEN COMPLETED. PACKLINKAGE PARAMETERS MUST BE THE SAME FOR ALL FAMILY MEMBERS.
C	GETHEAD AND FORGETHEAD: LOCATION OF A SPACE USED TO ALLOCATE DISK. PACKLIST: LIST OF AVAILABLE SPACE. AVAILLIST: COMPLEMENTS PACKLIST.
D	PACKINFO: LIST OF FAMILY MEMBERS.
E	LINKS: PACK SEGMENT NUMBER WHERE THE INDICATED TABLES ARE LOCATED. BACKUPLIST: ENTRY FOR EACH DUPLICATE DIRECTORY (OR MCP).
F	THESE LOCKS ONLY EXIST IN THE BASE UNIT OF THE FAMILY.
G	UBLOCKS: FILES THAT ARE INDEXED BY AN ASTERISK AND FILES THAT ARE INDEXED BY A USERCODE IN THE FAST DIRECTORY.
H	PACKTIMESTAMP: TIME AT WHICH THE LAST SPACE WAS ALLOCATED
I	USEGOLOC: LOCATION OF SEG [0] FOR THE PACK OR FAMILY FILE DIRECTORY.

Figure 5-4. Peripheral Unit Analysis (End)

IOCB

The IOCB command prints the Input/Output Control Block (IOCB) specified by <number>.

<iocb>

```
    -- IOCB ----------<number>--------|
          |      |              |
          |- AT -|              |
          |                     |
          |- VIA --<ASD number>-|
```

Semantics for the B 7900 and A 15 Systems:

The IOCB command analyzes both IOCBs and Hardware Control Blocks (HCBs) for the B 7900 and A 15 systems. The word located at the <number> is a descriptor that may refer to either an IOCB or an HCB. If the descriptor refers to an IOCB and an HCB has been allocated for that IOCB then both structures are analyzed. Word three of the structure pointed to by the descriptor is checked to determine if the the descriptor is for an HCB or an IOCB. If the tag of the word is 5 (indicating that it is a data descriptor), then the word is assumed to be the HCB Self Pointer and the descriptor is assumed to point to an HCB; otherwise, the descriptor is assumed to point to an IOCB.

Semantics for MCP/AS systems:

For systems using the MCP/AS, the IOCB command prints the IOCB specified by the <ASD number>.

IOTABLE (A SERIES MLIP SYSTEMS)

The IOTABLE command gives I/O information pertaining to the specified MLIP. This information includes expansion of the IOCB queue pointed to by the I/O table.

```
<iotable>

    -- IOTABLE --< MLIP number >--|
```

Semantics:

<MLIP number>

An integer between 0 and 7.

Example:

```
IOTABLE 2

##### I/O TABLE FOR MLIP# 2 @ 5 C00001 120347   #####
         I/O TABLE CW      :0 10C000 000000
         IOCB QUE HEAD PTR:5 C00000 3202F8
                                          ( IOCB QUEUE )
                                          IOCB QUE CW  :0 10C100 000000
                                          IOCB QUE HEAD:0 000000 000000
                                          IOCB QUE TAIL:0 000000 000000


         MLIP QUE CW       :0 10C200 000000
         MLIP QUE HEAD     :0 000000 000000
         MLIP QUE TAIL     :0 000000 000000
         EMP DEST. SET     :0 000000 000400 EMP#10.
         MLIP DEST. SET    :0 000000 000006 MLIP #1, MLIP#2.
         SCRATCH AREA      :5 E00001 C1D551
```

KEEP

The KEEP command causes the last command entered to be saved for future use. This command is stored in a 10-deep, first-in-first-out queue. Each entry has a number between 0 and 9, inclusive, which is used if the entry is recalled by the USE command. (Refer to "USE".)

<keep>

```
    -- KEEP ---------------|
            |          |
            |-<number>-|
```

Semantics:

KEEP

> The last command entered is saved at the next available number between 0 and 9. The command is retrieved by the USE command.

KEEP <number>

> Assigns a command to be stored at the specified number between 0 and 9.

See also
    USE 181

LIB

The LIB command causes library information to be analyzed and printed.

<lib>

```
    -- LIB --- AT --<simple address>-----|
            |                          |
            |- VIA --<ASD number>-----|
            |                          |
            |- MAP -------------------|
            |        |                 |
            |        |-<number>--------|
            |        |                 |
            |        |- ASN --<number>-|
            |        |                 |
            |        |- BOX --<number>-|
            |        |                 |
            |        |- GLOBAL --------|
            |                          |
            |- SL ---------------------|
```

Semantics:

LIB AT <simple address>
LIB VIA <ASD number>

Analyzes the contents of the indicated location of memory as a library structure.

LIB MAP

Displays the contents of the library map and the pointers for each of the address spaces into the map. Empty slots in the map are not displayed.

LIB MAP <number>

Displays the contents of a linked list chain in the library map, starting at the given index.

LIB MAP ASN <number>
LIB MAP BOX <number>
LIB MAP GLOBAL

Displays the contents of the linked list chain in the library map for the indicated address space.

LIB SL

Displays the system library function definitions.

Examples:

```
INPUT: LIB AT 45A8
----  HEADER  ----
    LEVEL = 3, LOCKED.
    STACK INFORMATION:  IMP AT 001D IN STK 090 = (*,0009).
----  USEINFO  ----
    LINKED TO STACKS: EXP AT 0019 IN STK 0A1.
----  AREAS  ----
    FREE         0031
    USEINFO      002D
    STACKREF     0006
    IMPORTS      000B
    EXPORTS      0000
    TYPES        000E
    NAMES        0012
    ATTRIBS      001A
----  IMPORT OBJECTS  ----
  ([V] = BY VALUE, [R] = BY REFERENCE, [N] = BY NAME)
    ALSOLINKED IS A INTEGER FUNCTION (1 PARAMETER);
        INTEGERIV];
        INDEX = 12, OBJECT = (*,000B).
----  ATTRIBUTES  ----
    VALUE = -5 H 400 0000 00005, INTNAME = L,
    TITLE = LIBRARY/MULTIPLELINKERROR.
----

INPUT: LIB MAP
LIB MAP ASN[0] = 2
LIB MAP ASN[1] = 6

MAP[1] = ASN, STK 05E, HDR 0055
MAP[2] = GLOBAL, STK 0A9, HDR 0057, NEXT = 3
MAP[3] = GLOBAL, STK 0A2, HDR 005A, NEXT = 1
MAP[4] = ASN, HDR, 0042
MAP[6] = RUN-UNIT, STK 083, HDR 0051, NEXT = 4

INPUT: LIB MAP GLOBAL
LIB MAP ASN[0] = 2

MAP[2] = GLOBAL, STK 0A9, HDR 0057, NEXT = 3
MAP[3] = GLOBAL, STK 0A2, HDR 005A, NEXT = 1
MAP[1] = ASN, STK 05E, HDR 0055)
```

DUMPANALYZER

```
INPUT: LIB SL
SL BNASUPPORT         = *36/SYSTEM/BNASUPPORT
    ONLY 1, LINKCLASS 1
SL COMSSUPPORT        = *SYSTEM/COMS
    LINKCLASS 1, TRUSTED, HDR 0057
SL DATACOMSUPPORT     = *36/SYSTEM/DATACOMSUPPORT
    ONLY 1, LINKCLASS 1, HDR 005A
SL GENERALSUPPORT     = *36/SYSTEM/GENERALSUPPORT
    HDR 0055
SL MCPSUPPORT         = ">> CURRENT MCP <<"
    MCPLIB, STK 010, HDR FFFF
```

LINKCHECK

The LINKCHECK command examines each memory link in the in-use and available areas of memory. If an error is found, an error message is displayed and DUMPANALYZER recovers and continues.

<linkcheck>

```
    -- LINKCHECK --|
```

AX (Accept) Command

```
    -- ? -- AX --- WHERE ----|
                 |         |
                 |- STOP --|
                 |         |
                 |- +T ----|
                 |         |
                 |- -T ----|
```

Semantics:

?AX WHERE

Asks where the LINKCHECK is.

?AX STOP

Stops the LINKCHECK and reprompts the user with a ":READY" message.

?AX +T
?AX -T

?AX +T allows the printing of the listing of text. ?AX -T suppresses the printing of the listing of text.

Example:

A typical response to the LINKCHECK command is as follows:

```
  LINKCHECK

  TAG ERR & 27621-27620  :  0  000000000000
```

LINKS

The LINKS command prints the addresses and contents of each link of the memory area that contains a specified <simple address>. The analysis also includes the following memory area attributes, which are encoded in the links:

a. Whether the area is available, save, csave (currently saved), or olay (overlayable).

b. If in-use, whether it is code, data or read-only.

c. The area size.

d. The area MOM address.

e. Whether or not the MOM address for the area is in the MCP D[0] stack and, if the name of that cell is available, the D[0] MOM address name.

f. The area stack number.

g. If present, the link C RCW.

h. The memory priority of the area.

i. If the area is available, the RCW trace of FORGETSPACE.

LINKS operates on only one memory subsystem in a tightly-coupled system. Use the BOX or ASN command to select the subsystem to be searched.

<links>

```
-- LINKS --<simple address>---------------|
                               |          |
                               |- EXPAND -|
```

Semantics:

The LINKS <simple address> EXPAND form of the command is valid only on systems using MCP/AS. It expands the fields of ASDs.

Examples:

The format of the information displayed by the LINKS command is given below. Formats are given for both in-use and available memory areas.

In-use area format on non-MCP/AS machines:

```
<area addr> <area type> : LENGTH=<length> MOM ADDR=<MOMADDRF>:<offset>

 LINK A : [<link addr>] 6 <link contents>
   D[0] MOM ADDRESS NAME : <name, if in MCP stack>
   TEMPSAVEF : <from Link A>
   PRECURSAVF : <from Link A>

 LINK B : <link address> 7 <link contents>
   MOM STACK : <STKNRF from Link B>
   OVERLAYCF : <from Link B>
   USAGEF :  <from Link B>

 LINK C : <link addr> 3 <link contents>
   LINK C RCW : <if present, the RCW in Link C>

 LINK Z : <link addr> 3 <link contents>

 MEMORY PRIORITY : <from link Z>
```

Available area format on non-MCP/AS machines:

```
<area addr> AVAILABLE AREA: LENGTH=<area length>

 AVAIL-A : [<link addr>] 1 <link contents>
 AVAIL-B : [<link addr>] 0 <link contents>
 AVAIL-Y : [<link addr>] 0 <link contents>
 AVAIL-Z : [<link addr>] 1 <link contents>

    RCW TRACE OF FORGETSPACE---
       STACK: <stack number>
       @<RCW> ( <line number> ) <procedure name>
       @<RCW> ( <line number> ) <procedure name>
       @<RCW> ( <line number> ) <procedure name>
                  .
                  .
                  .
```

In-use area format on MCP/AS systems:

```
<area addr> <area type> : LENGTH=<length> MOM ADDR=<MOMADDRF>:
                                                   <offset>
ASD(1): [<ASD1 addr>] TAG <ASD1 contents>
   ASD1_PRESENTTOPROCF   =  <content of ASD1_PRESENTTOPROCF>
   ASD1_NOTSTACKF        =  <content of ASD1_NOTSTACKF>
   ASD1_PRESENTFORIOF    =  <content of ASD1_PRESENTFORIOF>
                         .
                         .
                         .
ASD(2): [<ASD2 addr>] TAG <ASD2 contents>
   ASD2_SPACEUSAGEF      =  <content of ASD2_SPACEUSAGEF>
   ASD2_ABSENTADDRF      =  <content of ASD2_ABSENTADDRF>
   ASD2_SIZEF            =  <content of ASD2_SIZEF>

ASD(3): [<ASD3 addr>]    =  <ASD3 contents>
                         .
                         .
                         .
ASD(4): [<ASD4 addr>]    =  <ASD4 contents>
                         .
                         .
                         .
LINK A : [<link addr>] 6 <link contents>
   SPACE USAGE           = <spaceusage>
   INUSE_SEGSTATEF       = <content of INUSE_SEGSTATEF>
   INUSE_ASDINDEXF       = <content of INUSE_ASDINDEXF>
   INUSE_SIZEF           = <content of INUSE_SIZEF>

LINK C : [<link addr>] 3 <link contents>
   LINK C RCW : <if present, the RCW in Link C>

LINK Z : [<link addr>] 3 <link contents>
   INUSE_SIZEF           = <content of INUSE_SIZEF>
```

Available area format on MCP/AS systems:

```
<area addr> AVAILABLE AREA: LENGTH=<area length>

   AVAIL-A : [<link addr>] 1 <link contents>
     AVAIL_SIZEF         = <content of AVAIL_SIZEF>

   AVAIL-B : [<link addr>] 0 <link contents>

   AVAIL-C : [<link addr>] 0 <link contents>

   AVAIL-D : [<link addr>] 0 <link contents>

   AVAIL-E : [<link addr>] 0 <link contents>

   AVAIL-Z : [<link addr>] 1 <link contents>
     AVAIL_SIZEF         = <content of AVAIL_SIZEF>
     ROW TRACE OF FORGETSPACE---
        STACK: <stack number>
        @<RCW> ( <link number> ) <procedure name>
        @<RCW> ( <link number> ) <procedure name>
        @<RCW> ( <link number> ) <procedure name>
                    .
                    .
                    .
```

LOCKS

Locks or events are defined as being in one of two states: normal or abnormal. An abnormal event is one that is procured or has a stack waiting on it. All EVENTs and EVENT ARRAYs declared in the MCP outer block are considered, but only abnormal ones are listed. Hard locks are not reported.

```
<locks>

    -- LOCKS --|
```

Example:

```
INPUT: LOCKS

DO EVENTS (PROCURED OR WITH WAITERS)

3CB 2 000000 008490  2 000000 000000 DCPHEYU[0002]
        (WAITING: STK 049)
    2 000000 008480  2 000000 000000 DCPHEYU[0003]
        (WAITING: STK 048)
355 2 3637DF 000005  2 000000 000000 HEADERLOCKS[0066]
        (UNAVAILABLE, HAPPENED, PROCURED BY STK 363 in SEG 7DF)
262 2 000000 0001E0  2 000000 000000 ANABOLEVENT
```

WARNING

The MCP global events and event arrays are located from bindinfo data in the MCP code file. If the analyzed code file does not match the dumping MCP code file, the event analysis may be incorrect.

MASK

The MASK command is used to set or examine the mask register. The MASK command is used in conjunction with the PATTERN and SEARCH commands to search a dump for all words that contain a particular pattern of bits. The mask register modifies the pattern in the pattern register by masking certain bits in the pattern. The SEARCH command searches for all words which match that part of the pattern which is not masked. (Refer to the PATTERN and SEARCH commands for further information.)

<mask>

```
    -- MASK ---------------------|
            |                   |
            |-<simple value>-|
            |                   |
            |- ABSENTDESC ---|
            |                   |
            |- ABSENTCOPY ---|
            |                   |
            |- MOMDESC ------|
```

Semantics:

MASK

MASK specified by itself displays the contents of the mask register. The default for the MASK register is 7 FFFFFF FFFFFF.

MASK <simple value>

Places a mask that can be expressed as a given <simple value> into the mask register. A mask is a distribution of 1s and 0s in a 48-bit word and its 3-bit tag. The mask indicates which bits are significant and which are irrelevant within the current pattern set by the PATTERN command. All bits in a mask that have a value of 1 are significant digits in the pattern. All bits in a mask that have a value of 0 are disregarded in the pattern. The mask most recently specified is stored in the mask register.

The mask can include the 3- bit tag for a word. To specify a TAG, the concatenation form of simple value should be used; input would specify 12 hexadecimal digits "& <number> TAG". The tag value should be no larger than 7; however, if it is larger than 7, the tag value is placed in the tag using modulus 8. If no tag is specified, the tag is assumed to be 0.

Although the mask represents 48 binary bits and a tag value of three binary bits, it is displayed and is most often set in hexadecimal. Refer to the following diagram of a word that contains a mask value, with its hexadecimal equivalent stated below it:

```
        bit#|-----------------------------------------------------------|
 TAG     47| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  |3
|----|     |----|----|----|----|----|----|----|----|----|----|----|----|
| 1  |   46| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  |2
|----|     |----|----|----|----|----|----|----|----|----|----|----|----|
| 1  |   45| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  |1
|----|     |----|----|----|----|----|----|----|----|----|----|----|----|
| 1  |   44| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  |0
|----|     |----|----|----|----|----|----|----|----|----|----|----|----|

  7          0    0    0    0    0    0    0    0    F    F    F    F
```

To indicate that the last 16 bits of the pattern word and its tag are significant, the corresponding mask command would be MASK FFFF & TAG 7. (It is understood that there are eight leading zeros.) The mask register would display its contents as 7 000000 00FFFF.

The default for the MASK register is 7 FFFFFF FFFFFF; in this case, all bits including the three tag bits (represented by the 7) are set to one, and therefore are significant in the pattern register.

MASK ABSENTDESC

The value 7 C00000 000000 is placed into the mask register, in preparation to search for an absent mom descriptor. An absent mom descriptor references a data structure on disk.

MASK ABSENTCOPY

The value 7 C00000 000000 is placed into the mask register, in preparation to search for an absent copy descriptor. An absent copy descriptor references a data structure on disk.

MASK MOMDESC

The value 7 C00000 000000 is placed into the mask register, in preparation to search for a present mom descriptor. A present mom descriptor references a data structure in core.

Examples:

```
INPUT: MASK
MASK:  7 FFFFFF FFFFFF

INPUT: MASK ABSENTCOPY
MASK:  7 C00000 000000

INPUT: MASK MOMDESC
MASK:  7 C00000 000000

INPUT: MASK 123456789ABC
MASK:  0 123456 789ABC

INPUT: MASK 1324 & 5 TAG
MASK:  5 000000 001324
```

See also

PATTERN 138

SEARCH. 162

MD

The MD (Memory Dump) command dumps the contents of a group of addresses in memory with no analysis.

```
<md>

    -- MD --<multiple addresses>--|
```

Example:

A small subset of the raw dump analysis, which is output from the MD command follows:

```
          INPUT: MD C119D to C1200

          C119D/00000   0 C00040 190C6D   7 001200 080C7F    ...
          C11A4/00007   0 000000 000000   0 300023 000096    ...
          C11AB/0000E   0 400000 00CF62   0 000000 000074    ...
          C11B2/00015   0 000000 000000   0 000000 000000    ...
    A.    C11B9/0001C   0 000000 000000  THRU C11DB(0003E)   ...
          C11DC/0003F   1 4000A0 000000   6 C0001C 180C5A    ...
          C11E3/00046   3 B20695 B4AE42   3 4E601A AFB208    ...
          ...
```

ITEM	DESCRIPTION
A.	The only editing done is the recognition of repeated words. For example, in the block of zero operands, C11B9 and C11DB are the addresses of the first and last zero operand, in hex. 0001C and (0003E) represent relative addresses; in a raw dump, they are relative to the beginning address requested. They cycle every 4000 (hex) words. Each word is broken into tag, upper half and lower half.

MDCODE (HDU SYSTEMS)

The MDCODE (Memory Dump Code) command dumps the contents of a group of addresses in the code environment of memory. No analysis of the contents is done. If the ASN does not have a separate code environment, the MDCODE command gives the same result as the MD command.

```
<mdcode>

    -- MDCODE --<multiple addresses>--|
```

See also
MD. 125

MEM

The MEM (memory) command lists the memory modules present and the range of addresses contained in each module on the system when a dump was taken.

```
<mem>

    -- MEM --|
```

Example:

```
    INPUT: MEM

         MEMORY MODULE CONFIGURATION
         BOX 0: 32 (80000) - 63 (FFFFF)
         BOX 1: 0 (00000) - 31 (7FFFF)
         BOX 2: 0 (00000) - 31 (7FFFF)
         BOX 3: 0 (00000) - 31 (7FFFF)
         TOTAL MODS READY: 128
```

The following example shows the format for displaying the memory configuration on B 7900 and A 15 systems. The display is based on the memory granule "page", since the page is the smallest unit of memory that can be saved or readied on B 7900 and A 15 systems. A page is 128K words. The display is grouped according to ASN number with address ranges shown for each valid ASN. An example of the format follows:

```
    MEMORY CONFIGURATION
    GLOBAL: 4 PAGES, 0-3   ;  (00000) - (7FFFF)
    ASN  1: 3 PAGES, 4-5,7 ;  (80000) - (BFFFF), (E0000) - (FFFFF)
    ASN  2: 4 PAGES, 4-7   ;  (80000) - (FFFFF)
    ASN 46: 1 PAGE , 4     ;  (80000) - (9FFFF)
    TOTAL PAGES READY : 12
```

MIX

The MIX command displays the contents of the task stack or segment dictionary stack associated with the supplied mix number. The results are the same as for the STACK command.

<mix>

```
    -- MIX --<decimal number>-----------|
                             |      |
                             |- SD -|
```

Semantics:

MIX <decimal number>

The contents of the task stack associated with the supplied mix number (<decimal number>) are displayed.

MIX <decimal number> SD

The contents of the segment dictionary stack associated with the supplied mix number are displayed.

See also
STACK 168

MODE

The MODE command allows the user to control the mode in which words are expanded in the ALLSTACKS, MIX, and STACK commands. It is also used to control how <simple value>s are expanded in the PV command. (Refer to the PV command.)

<mode>

```
-- MODE --- ? --------------------|
        |                      |
        |- + ---<mode option>-|
        |     |
        |- - -|
```

<mode option>

```
---- ALL ------|
  |          |
  |- ARRAY -|
  |          |
  |- BCL ---|
  |          |
  |- CODE --|
  |          |
  |- DEC ---|
  |          |
  |- EBC ---|
  |          |
  |- FIB ---|
  |          |
  |- FILE --|
  |          |
  |- IOCB --|
  |          |
  |- LIB ---|
  |          |
  |- LOCK --|
  |          |
  |- OCT ---|
  |          |
  |- PCW ---|
  |          |
  |- PIB ---|
  |          |
  |- SIB ---|
```

Semantics:

MODE?

The current mode(s) is displayed.

MODE+ <mode option>

The specified mode is added.

MODE- <mode option>

The specified mode is deleted.

ALL

The ALL modifier is inclusive of all of the following modes.

ARRAY

If ARRAY mode is used, mom descriptors are expanded subject to the limit established by the ARRAYLIMIT command. If the array is multidimensional, each dimension is expanded as it is encountered. Information concerning the current level is printed in the left margin when multidimensional arrays are expanded.

BCL

Burroughs Common Language. The word is translated into Burroughs Common Language characters.

CODE

If the CODE option is used, DUMPANALYZER expands and lists code areas and read-only data areas. When analyzing a stack, the code for return control words (RCWs) is expanded into mnemonics for the operators and names of items at the D[0] level.

Op codes for all systems are recognized. An op code that is not defined for any of the systems is displayed in the form m*hh, where hh is the code syllable in hexadecimal and m is a letter indicating the context in which the syllable was encountered:

P: Primary (normal context)

V: Variant (right after code 95)

E: Edit (right after EXSD, EXSU, EXPU)

A: Vector(Array) = (in the word after VMES, or between VMEM and either VMEX or VEBR)

DEC

Decimal. The word is translated into a decimal number.

EBC

EBCDIC. The word is translated into Extended Binary Coded Decimal Interchange Code (EBCDIC) characters.

FIB

Expansion of FIBs in the stack is controlled by the FIB option. Buffers are dumped and analyzed.

FILE

FIB and FILE are synonyms.

IOCB

If IOCB mode is used, descriptors in the stack referencing an IOCB are expanded.

LIB

If the LIB option is used, library structures are expanded and analyzed. This applies to both library templates and library directories. (Refer to the LIB command.)

LOCK

If the LOCK option is used, the specified value is analyzed to determine whether it is a "hard lock" (tag of 0) or a "soft lock" (tag of 2).

OCT

Octal. The word is translated into an octal number.

PIB

If PIB mode is used, arrays that have an unusual field of PIB marks in their memory links are displayed as PIBs.

PCW

Program Control Word. A word is analyzed as if it were a program control word. The expansion of PCWs in stacks is optional; normally, PCWs are not expanded, but they are expanded if PCW is set.

SIB

If SIB mode is used, SIB descriptors are expanded. Any SIB descriptor found in the dumping stack will be analyzed as a SIB stack.

See also

PV. 150
LIB 113

MSCW

The MSCW command helps to analyze stacks that have a corrupt Mark Stack Control Word (MSCW) or are in a state in which DUMPANALYZER cannot analyze them correctly.

<mscw>

```
-- MSCW -- FOR --<number>--- AT --<number>----|
                             |                |
                             |- ? ------------|
```

Semantics:

MSCW FOR <number> AT <number>

The first <number> is the number of the specified stack. The second <number> is an offset, which must be the location of a valid MSCW in the stack. All MSCWs below this MSCW must also be valid. The MSCW indicated by <number> is marked off in the stack, so that when the stack is examined the following message appears in the MSCW's place:

```
MSCW ASSUMED AT THIS POINT BY USER SPECIFICATION
```

An MSCW is reset by setting the second <number> to zero; that is, the offset is zero.

MSCW FOR <number> ?

The setting of the MSCW for the specified stack is interrogated.

NAMES

The NAMES command causes the entire list of MCP names and addresses to be printed. SORT intrinsics are invoked to sort the names alphabetically. (See Figure 5-5.)

```
<names>

    -- NAMES --|
INPUT: NAMES

                         IDENTIFIER-ORDERED MCP STACK ADDRESSES FOR *SYSTEM/MCP32252A/FMLYINX002

PROCEDURE ADDRESSES ARE SHOWN AS PCW/SEGMENT

0585      ABCOUNT                     /08E0 ABNORMALTERMINATIONMESSER                                    0598/0B40 ABORTJOBLOG
0104      ABOXINFO               0583      ABQLOCK                  00A4/06BE ACCEPTAMSG                 006A/06BD ACCEPT
058C      ACQCNT                 0587      ACQSTOPPER               0589      ACQ                        00EC/0608 ACQUIREUNITS
00FF      ACQUIREUNITWORD        058E      ACRUNNING                0439/093A ADDALOGICALUNIT            064D/064E ADDASWAPDISK
0234/0695 ADDGRAPHEDGE           01A8/01CE ADDSTACKSEARCHEDGE       066C/066B ADDSWAPCORE                053B      ADHOCCOUNT
053A      ADHOCTIME              037E      ADLOCK                   0218      AEVPIB                     0131/062A AITINSERT
0311/0758 ALIENIOHANDLER              /0701 ALIENIOHANDLER              /0758 ALIENIOHANDLER             0512/0AC5 ALLOCATEEVENT
051A/0ACD ALLOCATELIST           0517/0ACA ALLOCATESPACE            0370/0827 ALLOCATE                   0640      ALLOCTABLE
0640      ALLOCTABLEV            01AA/01CE ALTERSEARCHABLE          007D/062E AMNESIA                    0262      ANABOLEVENT
0268      ANABOLHEAD             0244/06A0 ANABOLISM                01F3      ANSWEREDNUMBERS            01E4/01F5 ANSWER
01F0      ANSWERING              058A      APPLE                    058B      APQCNT                     0586      APQSTOPPER
0588      APQ                    058D      APRUNNING                038A      ARCHIVEEVENT               0389/0872 ARCHIVEHANDLER
038F      ARCHIVEINFO            038C      ARCHIVELOCK              038E      ARCHIVETIMESTAMP           0163      AREAMANAGERSNR
0158/0638 AREAMANAGER            0007/062C ARRAYDEC                 0260      ARRAYEXTSPIB               00D5/0AD5 ARRAYSEARCHP
00CB      ARRAYSPIB              0002      ARRAYSTACK               00B3      ASCTOBCL                   00B4      ASCTOEBC
00B2      ASCTOHEX               03DA      ATTACHEDLINE             059F/084F ATTACHSPOQ                 04DC      ATTEMPTINGNETPLUS

                         ADDRESS-ORDERED MCP-STACK IDENTIFIERS FOR *SYSTEM/MCP32252A/FMLYINX002

/ DENOTES CODE SEGMENT

/0001 EATEIGHTEENSYLLABLES,ETC    0002 STACK                   0003 D03                      0004 WORK
/0005 D03                         0007 ARRAYDEC                0008 CCDOVARIABLEPPB          0009 LINEINFODESC
 000A BLOCKEXIT                   000B GOTOSOLVER              000C DABMEM                   000D TIMETUNNEL
 000E SOFTWAREINTERRUPTATTACH     000F NEWDOPEVECTOR           0010 PICKASTACK               0011 INTRINSICINFO
 0012 MYJOB                       0013 HOLD                    0014 MEMDUMP                  0016 FREEZELIB
 0017 PROGRAMDUMP                 0018 TIMEINTRINSIC           0019 MACHINEID                001A DMSUPDATEDISKHEADER
 001B CLOSE                       001C PIO                     001D TIMEOFDAYWORDV           001E VOLUMELIST
 001F OPEN                        0020 FAULTDEC                0021 BCLTOEBC                 0022 EBCTOBCL
 0023 POTL                        0024 POTM                    0025 POTH                     0026 TRUTHSETS
 0027 HEXTOBCL                    0028 MESSER                  0029 ATTRIBUTEGRABBER         002A ATTRIBUTEHANDLER
 002B GETSTATUS                   002C SETSTATUS               002D SYSTEMSTATUS             002E USERIOERROR
 002F SWAP                        0030 WRITESPO                0031 CCDOSTRINGCONV           0032 LOADCONTROL
 0033 FORGETSPACE                 0034 CCDOSTRINGFUNCTION      0035 WAITIO                   0036 NOTIOREQUEST
 0038 FORGETAREA                  0039 FAULTHANDLER            003A PROCESSKILL              003B SPACEOF
 003C SPACEN                      003D MOMTOVECTOR             003E TURNOVERLAYKEY           003F MAKEPRESENTANDSAVE
 0040 RELEASEHEADER               0041 DISPLAYTOSTANDARD       0042 GETSPACE                 0043 DISKWAIT
```

WHEN A TABLE OF NAMES AND ADDRESSES IS PRODUCED, EITHER NON-INTERACTIVELY OR IN RESPONSE TO THE NAMES COMMAND, THE SORT INTRINSICS ARE INVOKED TO SORT THE NAMES ALPHABETICALLY . WHILE THIS SORT IS TAKING PLACE, THE HI RESPONSE IS "SORTING MCP NAMES".

Figure 5-5. Identifier and Address-Couple Ordered Stack Listing

OLAYINFO

The OLAYINFO command analyzes overlay file allocation. This command is used when overlay file corruption is suspected.

<olayinfo>

```
    -- OLAYINFO --------------------------|
                 |                     |
                 |-<stack>-------------|
                           |           |
                           |-<olayadd>-|
```

Semantics:

OLAYINFO

DUMPANALYZER finds all descriptors to present or absent overlayable data, checks for overlapping allocation or other errors, and compares the descriptors with OLAYINFODESC bit vectors. All stacks are analyzed.

OLAYINFO <stack>

If <stack> is present, only the specified stack is analyzed.

OLAYADD <stack> <olayadd>

If <olayadd> is present, overlay allocations neighboring <olayadd> are shown.

OPT

The OPT command lists the compile-time and run-time option settings in effect when the dump was taken.

```
<opt>

    -- OPT --|
```

Example:

```
INPUT: OPT

OPTIONS: SET: TERMINATE, NOCHECK, LPBDONLY, AUTORM, AUTORECOVERY,
              TRANSWARNINGS, AUTODC, CPBDONLY, NEWPERETRY,
              SERIALNUMBER, CONTROLOLDWFL, BNARECOVERY

        RESET: OPEN, DIAGNOSTICS, CDONLY, DUPSUPERVISOR, NODUMP,
              DUPINTRINSICS, CRUNCH, BACKUPBYJOBNR, FULLTRANSLATION,
              NOFETCH, RESOURCECHECK, NOSUMMARY, DIRDEBUG,
              CATALOGING, OKTIMEANDDATE, LOGPOSITIONING, ARCHIVING,
              LOCKTRACE, IORANGECHECK, SWAPALLJOBS, NORVRSPAPERTAPE,
              MIRRORING, DIAGNOSTICDUMP, AUDIT, FILESATURATION,
              EOTSTATISTICS, PATHBALANCING, GMMDEBUG, ISCDEBUG,
              IODIAGNOSTICS, PORTDEBUG, USECATDEFAULT, CATTEST,
              MCPTEST

                       COMPILETIME OPTIONS SET
                             DIAGNOSTICS
                             EXPERIMENTAL
                             LINEINFO
                             LOCKTRACE
                             READLOCK
                             READLOCKTIMEOUT
                             RESTART
                             SWAPALL
                             TRACE
                             CATALOGLEVEL = 0

         MODULE ALTERNATIVES SELECTED: PORTS_BNA_V2
                                       INITFILE_BOOTCODE
                                       DIALSEARCH
                                       CONFIGCONTROL60
                                       B6XXXPROCCONTROL
                                       HDPMAINTENANCE
                                       DCPHDPSERVICES
                                       NIFHDPCONTROL
                                       PHYSICALIOHDP
```

PATTERN

The PATTERN command is used to set or examine the pattern register.

The PATTERN command can be used with the SEARCH command to search for all words which contain a pattern involving only a subset of the bits within a word and its tag. In this case, the MASK command must be used to mask off some of the bits in the pattern as non-significant. (Refer to the MASK and SEARCH commands for further information.)

<pattern>

```
    -- PATTERN ---------------------|
                |                  |
                |-<simple value>-|
```

Semantics:

PATTERN

If no <simple value> follows PATTERN, the contents of the PATTERN register are displayed.

PATTERN <simple value>

The pattern register is loaded with <simple value>, which may then be used as a nonvolatile search pattern.

A pattern is a distribution of 1s and 0s in a 48-bit word and its 3-bit tag. This pattern of ones and zeros is used by the SEARCH command to search the dump for all words that match the given pattern, bit for bit. The pattern most recently specified is stored in the pattern register.

The pattern includes the 3-bit tag for a word. To specify a TAG, the concatenation form of simple value is used - input would specify 12 hexadecimal digits "& <number> TAG". The tag value should be no larger than 7; however, if it is larger than 7, the tag value is placed in the tag using modulus eight. If no tag is specified, the tag is assumed to be 0.

Although the pattern represents 48 binary bits and a tag value of three binary bits, the pattern is displayed and is most often set in hexadecimal. A diagram of a word that constitutes a pattern is given, with its hexadecimal equivalent stated below it, as follows:

```
        bit#|-----------------------------------------------------------|
 TAG     47| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  |3
|----|     |----|----|----|----|----|----|----|----|----|----|----|----|
| 1  |   46| 0  | 0  | 0  | 0  | 1  | 1  | 0  | 0  | 1  | 1  | 1  | 1  |2
|----|     |----|----|----|----|----|----|----|----|----|----|----|----|
| 1  |   45| 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  | 1  | 1  |1
|----|     |----|----|----|----|----|----|----|----|----|----|----|----|
| 1  |   44| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  |0
|----|     |----|----|----|----|----|----|----|----|----|----|----|----|

  7          0    0    0    0    4    4    2    2    F    F    F    F
```

To put the pattern of bits shown in the word above into the pattern register, the corresponding pattern command would be PATTERN 4422FFFF & 7 TAG (it is understood that there are four leading zeros). The pattern register would display its contents as 7 000044 22FFFF.

The default for the PATTERN register is displayed as 0 000000 000000.

Examples:

```
INPUT:    PATTERN
PATTERN: 0 000000 000000

INPUT:    PATTERN 404040404040
PATTERN: 0 404040 404040

INPUT:    PATTERN 123456789ABC & DEC 10 TAG
PATTERN: 2 123456 789ABC
```

See also
MASK. 122
SEARCH. 162

PC or PRINTCODE

The PC or PRINTCODE command translates code words into their mnemonic equivalents and prints the results.

```
<pc>

    ---- PC ----------<simple address>--------------------------------->
      |             |                  |           | |               |
      |- PRINTCODE -|                  |- : --<PWI>-| |-<until part>-|

    >-------------------------------------------------------------------|
      |                 |
      |- LL --<lex level>-|
```

Semantics:

Translation of code begins on or before the designated <simple address>, which is the segment base. Translation of code ceases when both of the following conditions are true:

1) The length specified is achieved.
2) The instruction word is complete.

<PWI> is a number that specifies a location relative to the segment base. The default value for <PWI> is 0.

If no length (<until part>) is specified, the default length is three words.

The <lex level>, a number, indicates the running environment level. If it is not specified, a lex level of 3 is assumed. If the lex level of the running environment is not 3, the lex level must be specified to ensure proper interpretation of operators that contain address couples, such as NAMC and VALC.

Non-tag-three words are translated, but a warning is also displayed.

All display headings (such as, in the example, the lines from "CODE ANALYSIS:" to the underline) are suppressed from remote output but appear in the printer output.

Example:

```
PC 85EE5:2B FOR 4 LL 1


     CODE ANALYSIS:
                    SEGMENT BASE: 85EE5
                    WORD OFFSET : 0002B
                    LENGTH      : 00004
                    LEX LEVEL   : 1

     PWI:PSI MNEMONICS                HEX CODE
     ---------------------------------------------
     002B:0  ZERO                     B0
     002B:1  ZERO                     B0
     002B:2  PUSH                     B4
     002B:3  MPCW 000600090BC3        BFFFFF000600090BC3
     002D:0  ISOL 15:48               9A0F30                ** NOT TAG-3 **
     002D:3  NAMC 00,01B3             41B3  GETFORGET
     002D:5  RDLK                     95BA
     002E:1  DUPL                     B7
     002E:2  BRFL 0005:3              A06005
     002E:5  LT8  1A(26)              B21A
```

See also

SIMPLE ADDRESS. 57

MULTIPLE ADDRESSES. 62

PIB

The PIB command prints the contents of a program information block (PIB). A PIB is a task variable associated with a process stack. The descriptor for active PIBs is in a SPIBVECTOR parallel to STACKVECTOR.

```
<pib>

    -- PIB ---<number>------------------|
             |                          |
             |- AT --<simple address>-|
```

Semantics:

PIB <number>

The PIB associated with the stack specified by <number> is printed.

PIB AT <simple address>

The PIB located at memory address <simple address> is printed.

PORT

The PORT command causes an analysis of a port file to be printed. A system response of "PORT OVERLAYED" implies that the port file was not in memory when the dump was taken. (See Figure 5-6.)

<port>

```
    -- PORT ---<port index>-------|
             |                |
             |- AT --<address>-|
```

Semantics:

Port <port index>

The <port index> is stored in the LIBRARYINFO word of a FIB using the port. The port with the given index is analyzed.

Port AT <address>

The port at the specified address is analyzed.

```
INPUT: PORT 6

PORT ATTRIBUTE BLOCK DESCRIPTOR: 5 800000 DOCAD3
00(00) 0 D7D6D9 E30006 MYPORTADDRESS               =6(6)
01(01) 0 C00007 001000 CONNECT                     =PEER
                       SECURITYTYPE                =PRIVATE
                       MAXSUBFILES                 =7(7)
                       CENSUS                      =0(0)
02(02) 0 000200 010000 ACTIVESUBPORTS              =2(2)
                       READABLESUBPORTS            =1(1)
                       DYINGSUBPORTS               =0(0)
03(03) 0 078000 030000 MAXRECSIZE                  =1920(780)
                       CHANGEDSUBFILE              =3(3)
                       LASTSUBFILE                 =0(0)
04(04) 0 000A00 080009 CHANGEEVENT                 =10(A)
                            2 000000 000000
                            2 000000 000001  EVENT ANALYSIS: (HAPPENED)
                       PORTLOCK                     =8(8)
                            0 000000 000000
                            0 000000 000000  EVENT ANALYSIS:
                       INPUTEVENT                   =9(9)
                            0 000000 000000
                            0 000000 000000  EVENT ANALYSIS:
05(05) 0 000700 080000 INTEVENT                     =11(B)
                            2 000000 000000
                            2 000000 000001  EVENT ANALYSIS: (HAPPENED)
                       PORTPTRNUM                  =7(7)
                       TITLE                       ="F
                       MYNAME                      =""
                       SECURITYGUARD               =""
06(06) 0 000000 00000A SUBFILE 1
       SUBPORT ATTRIBUTE BLOCK DESCRIPTOR: 5 800001 A42598
       00(00) 0 E2E4C2 D7000A MY SUBPORTADDRESS       =10(A)
       01(01) 0 E00500 010006 INCHANGELIST            =TRUE
           .
           .
           .
08(08) 0 000000 00000D SUBFILE 3
       SUBPORT ATTRIBUTE BLOCK DESCRIPTOR: 5 800001 A4201A
       00(00) 0 E2E4C2 D7000D MYSUBPORTADDRESS        =13(D)
       01(01) 0 800500 030006 INCHANGELIST            =TRUE
                              INSABLIST               =FALSE
                              COUNTED                 =FALSE
                              ABORTED                 =FALSE
                              COMPRESSING             =FALSE
                              COMPRESSIONPOSSIBLE     =FALSE
                              RECEIVINGCOMPRESSED     =FALSE
                              YOURUSERCODEISNULL      =TRUE
                              HOSTNAMEWASNULL         =FALSE
                              YOURNAMEWASNULL         =TRUE
                              MYSUBFILEINDEX          =3(3)
                              MYPORTINDEX             =6(6)
       02(02) 0 30000F 000000 SUBFILESTATE            =OFFERED(OPENPENDING)
                              SUBFILEERROR            =NO ERROR
                              WAITCOUNT               =0(0)
                              DIALOGPROTOCOLLEVEL     =0(0)
                              BALLOT
            CANDIDATE DESCRIPTOR: 5 800009 141F34
            00(00) 0 C3C1D5 C4000F CANSELF                         =15(F)
            01(01) 0 9A8100 C00000 CDONE                           =TRUE
                                   NEWCAN                          =FALSE
                                   CINLIMBO                        =FALSE
                                   CLOCAL                          =TRUE
                                   CCOMPRESSION                    =TRUE
                                   CAGREEDCOMPRESSION              =FALSE
                                   CINLIST                         =TRUE
                                   CYOURHOSTNAMEWASNULL            =FALSE
           .
           .
           .
09(09) 0 000000 000000 SUBFILES 4-7 UNUSED
```

Figure 5-6. Port, Subport and Candidate Analysis

PRINTER

The PRINTER command routes output to the line printer instead of the terminal. The standard DUMPANALYZER header page is printed as a preamble. Each command that causes the information to be dumped is also printed.

```
<printer>

    -- PRINTER --|
```

PRINTVAL

This command is a synonym for the PV (or PRINTVAL) command. Refer to the PV command.

See also
PV. 150

PROC (MLIP AND MPX SYSTEMS)

The PROC command controls the resolution of addresses not in Global Memory. All non-Global addresses are interpreted as being in the processor specified by <number>. The PROC command with no modifier displays the current processor and box setting or the message NOT TIGHTLY COUPLED. When using ODT operation, the system response is displayed on the ODT. PROC is not valid on B 7000 Series systems; instead, the ASN command should be used to access the memory of a local ASN, both for tightly-coupled and extended memory systems.

<proc>

```
    -- PROC ---------------|
             |           |
             |-<number>-|
```

See also

ASN 79

PROCS (A SERIES MLIP SYSTEMS)

The PROCS command causes all processors that were running on the system (both E-mode and MLIP) to be displayed.

```
<procs>

    -- PROCS --|
```

Example:

```
    PROCS

    PROCESSORS CURRENTLY ON THE SYSTEM ARE:
    MLIP #1
    MLIP #2
    EMP  #9  IN STACK 036  S=1E4EE
    EMP  #10 IN STACK 011  S=1FE55
```

PROCSTACKS

The PROCSTACKS command displays the contents of each stack that has a processor currently on it. The stack that requested the memory dump is also displayed. The contents of the stacks are formatted and interpreted before they are displayed.

```
<procstacks>

    -- PROCSTACKS --|
```

PV

The PV or PRINTVAL command displays the specified <simple value> in several different possible forms.

<pv>

```
                          |<-------------------|
                          |                    |
-- PV --<simple value>-------------------------|
                            |                |
                            |-<mode option>--|
                            |                |
                            |- MODE ---------|
                            |                |
                            |- RCW ----------|
                            |                |
                            |- EBC -- ARRAY -|
```

Semantics:

PV <simple value>

The specified <simple value> is displayed in hexadecimal form.

PV <simple value> <mode option>

The specified <simple value> is displayed in the desired <mode option>. (Refer to the MODE command.)

PV <simple value> MODE

Displays the specified <simple value> in the modes that are currently set. (Refer to the MODE command.)

In the FIB mode, <simple value> is expanded as a FIB. <simple value> must be a present descriptor. If the length field is incorrect, a warning message indicating that the value is probably not a FIB is displayed.

In the IOCB mode, the IOCB word in the base of the stack and any descriptor may be analyzed as an IOCB.

On B 7900 or A 15 systems, the PV command in the IOCB mode analyzes both IOCBs and HCBs. The word whose contents are <simple value> is a descriptor that may refer to either an IOCB or an HCB. If the descriptor refers to an IOCB and an HCB has been allocated for that

IOCB then both structures are analyzed. Word three of the structure pointed to by the descriptor is checked to determine if the descriptor is for an HCB or an IOCB. If the tag of the word is 5 (indicating that it is a data descriptor), then the word is assumed to be the HCB Self Pointer and the descriptor is assumed to point to an HCB; otherwise, the descriptor is assumed to point to an IOCB.

If the MCP option READLOCK is set, each hard lock is shown with sequence number and procedure name in the LOCK mode.

In the construct "PV M[a] . . .", if M[a] and M[a + 1] are both Tag-2 words, they are analyzed together as a double operand. This construct is effective in DEC, OCT, BCL, EBC, or LOCK modes.

PV <simple value> RCW

In the RCW mode, <simple value> is analyzed as if it were an RCW.

PV <simple value> EBC ARRAY

In the EBC ARRAY mode, an array is displayed in both hexadecimal and EBCDIC formats.

See also
MODE. 129

QUEUE

The QUEUE command displays DCALGOL queues. QUE is an acceptable abbreviation.

<queue>

```
    -- QUEUE -------------------------|
              |          | |        |
              |-<number>-| |- MSG -|
```

QUEUE

If QUEUE is specified with no <number>, all DCALGOL queues are printed.

QUEUE <number>

If <number> is specified, the indicated queue is printed.

MSG

If MSG is specified, an analysis of messages in the queues is also printed.

Example:

The following is an example of the results returned when "QUEUE 157" is specified:

```
 INPUT: QUEUE 157

 DUMP OF DCALGOL QUEUE STACK (3EE) : 5 800016 1A563D

 QUEUE
 NUMBER
 ------
 0157    5 800000 D92354 (QUEUE ADDR = 92354)
        /
   A.---
         0000  0 00863E 0979F1 QLINKAGE (QTAIL=863ED,QHEAD=979F1)
         0001  0 000821 570001 QINFO  (ACTIVATING SNR=082, USERS=1)
         0002  0 000023 00002B QSIZE  (TOTAL SIZE=35, MEMORY SIZE=43)
   B.    0003  0 000000 080154 QMSGINFO (MSGCOUNT=8,MEMORYLIMIT=340)****
         0004  0 000000 0A005A QTANKINFO (ROWSIZE=10,BLOCKSIZE=900)
         0005  0 000000 000000 QBUFFDESC
         0006  0 000000 000000 QLOCKEVENT
         0007  0 000000 000000
         0008  2 000000 000001 QINSERTEVENT
         0009  2 000000 000000
   C.    000A  0 000000 000000 QTIBDESC
         000B  0 000000 000000 QLOCKCONTEND
         000C  0 000000 000000 QLOCKOWNER
```

ITEM	DESCRIPTION
A.	The queue stack is a non-running stack of descriptors referencing the actual queue.
B.	The asterisks at the end of this entry imply that there are messages to be removed from the queue. MSGCOUNT, here, indicates that there are eight messages to be removed.
C.	QTIBDESC: Queue task information block.

RECESS

The RECESS command allows the user to exit from DUMPANALYZER without removing the "pseudorecovery" file created during DUMPANALYZER's initialization process. For information on the usage of the pseudorecovery file, refer to "DUMPANALYZER FILES" in this section.

```
<recess>

    -- RECESS --|
```

See also
    DUMPANALYZER FILES. 48

RELEASE

The RELEASE command closes the current line printer file and opens a new one.

```
<release>

    -- RELEASE --|
```

RELX

The RELX command causes the current line printer file to be printed while the program is still running. When the line printer file is printed, it is preceded by a heading page containing the usercode and mix number of the DUMPANALYZER task, as well as the title "ANALYZEDUMP". RELX is only valid when DUMPANALYZER is run in PRINTER mode.

```
<relx>

    -- RELX --|
```

REMOTE

The REMOTE command routes output to the terminal rather than to the line printer. REMOTE is not valid when DUMPANALYZER is run from the ODT.

```
<remote>

    -- REMOTE --|
```

REPEAT

The REPEAT command causes the previous interactive command to be repeated. The output from the command is routed to either the printer or the remote terminal.

<repeat>

```
    -- REPEAT -----------------------|
                |                    |
                |-------- PRINTER -|
                |      |
                |- TO -|
```

Semantics:

REPEAT

The previous command is repeated. Output is repeated to whatever peripheral device has been initially specified.

REPEAT PRINTER
REPEAT TO PRINTER

The previous command is repeated and the output is directed to the line printer. If REMOTE operation is in effect, each REPEAT to the printer is in a separate printer backup file. This version of the REPEAT command can be used to obtain a hard copy of command output during an interactive session at a remote terminal.

RESULTQ (HDU AND MLIP SYSTEMS)

The RESULTQ command invokes output of the result queues generated by I/O operations. These result queues display the linking together of IOCBs after the I/O operations are complete. This command is only valid on MLIP and HDU systems.

```
<resultq>

    -- RESULTQ --|
```

RJ

The RJ or RIGHTJUST command enables (sets) or disables (resets) right justification of terminal output. The default state is right justification enabled. When right justification is enabled, terminal output that exceeds the terminal width is displayed right-justified on the succeeding line. When right justification is disabled, this excess will be left-justified on the next line.

<rj>

```
    ---- RJ --------------------|
      |            | |      |
      |- RIGHTJUST -| |- + -|
                     |      |
                     |- - -|
                     |      |
                     |- ? -|
```

Semantics:

RJ
RJ +

    Enables right justification.

RJ -

    Disables right justification.

RJ ?

    Shows the current state of right justification.

Examples:

```
    INPUT : RJ
    ** RIGHT JUSTIFICATION IS SET **

    INPUT : RJ -
    ** RIGHT JUSTIFICATION IS RESET **

    INPUT : RJ ?
    ** RIGHT JUSTIFICATION IS RESET **
```

SAVE

The SAVE command saves the contents of memory dump tapes and relevant information from the MCP code file in a disk file for later analysis. The file is saved under the <file title> given; <file title> must appear in quotes. LINEINFO must be set and the MCP global names must be available. The AREASIZE of the saved file is 10 records or 500 segments.

The file created by the SAVE command can be quite large (from 30,000 to 150,000 disk segments). It may be desirable to copy the file to tape rather than leaving it on disk for this reason.

The file created using the SAVE command is not removed when DUMPANALYZER is exited. It must be removed using CANDE commands or MARC screen functions.

```
<save>

    -- SAVE -- " --<file title>-- " --|
```

SEARCH

The SEARCH command, when used in conjunction with the MASK and PATTERN commands, provides the ability to check each word in a memory dump for a specified pattern of bits; this pattern may include all 48 bits within the word and its three tag bits, or it may search for all words that have a subset of those 48 bits in common. SEARCH can cause a search for a pattern that is specified by the PATTERN command and limited by the mask set in the MASK command. Patterns for a search may also be specified within the SEARCH command; these patterns do not use the mask and pattern registers.

The addresses of the words that match the specified pattern, along with the contents of those words, can be saved in auxiliary storage so that they can be retrieved as desired. The saved words may be searched again with different search patterns.

The SEARCH command operates on only one memory subsystem in a tightly-coupled system. The BOX or ASN command is used to select the subsystem to be searched. (Refer to the BOX and ASN commands.)

<search>

```
    -- SEARCH ---------------------------------------------------------->
                |              |
                |- ABSENTCOPY -|
                |              |
                |- ABSENTDESC -|
                |              |
                |- MOMDESC ----|
                |              |
                |- PREVIOUS ---|

       |<--------------------------------------|
       |                                       |
    >-------------------------------------------------------------------|
         |                                   |   |                 |
         |-/1\- REFS --<simple address>------|   |     |<--------| |
         |                                   |   |     |         | |
         |-/1\- RANGE --<multiple addresses>-|   |- : ----- T -----|
         |                                   |           |     |
         |-/1\- MASK --<simple value>--------|           |- K -|
         |                                   |
         |-/1\- PATTERN --<simple value>-----|
-- ?AX --- WHERE --------|
        |           |
        |- STOP ------|
        |           |
        |- - -- TEXT -|
        |           |
        |- + -- TEXT -|
        |           |
        |- - -- LIST -|
        |           |
        |- + -- LIST -|
        |           |
        |- - -- KEEP -|
        |           |
        |- SKIP ------|
```

Semantics:

SEARCH

If SEARCH is used without any modifying tokens, the contents of the pattern and mask registers are used in the search. Refer to the MASK and PATTERN commands for information about how these registers are set.

ABSENTCOPY

ABSENTCOPY causes a search for all absent copy descriptors. An absent copy descriptor references a data structure on disk.

ABSENTDESC

ABSENTDESC causes a search for all absent mom descriptors. An absent mom descriptor references a data structure on disk.

MOMDESC

MOMDESC causes a search for all present mom descriptors. A present mom descriptor references a data structure in core.

PREVIOUS

PREVIOUS causes a search from among a group selected from a previous SEARCH command and stored via the K option described below.

REFS <simple address>

REFS causes a search for present descriptors that reference the section of memory including the specified address.

RANGE <multiple addresses>

RANGE <multiple addresses> indicates a range of memory over which to search. <multiple addresses> indicates which set or sets of addresses constitute the range. The syntax for <multiple addresses> is provided in "Basic Constructs".

MASK <simple value>

MASK <simple value> may be used to specify a mask to be used for the duration of the search only, but not to be placed in the mask register. The contents of the mask is <simple value>, a hexadecimal representation of a 48-bit word with an optional tag, which places a 1 in each bit that is significant in the pattern, and a 0 to mask all insignificant bits.

PATTERN <simple value>

PATTERN may be used to specify a pattern for which to search; however, this pattern will not be placed in the pattern register. The contents of the pattern is <simple value>, a hexadecimal representation of a 48-bit word with an optional tag, whose significant bits are pointed to by the mask. The SEARCH command takes this pattern and finds words which match the pattern in the significant bits which have been selected by the mask.

:T

":T" lists the words that match the current pattern modified by the current mask and the hexadecimal 5-digit address for each word next to its contents.

:K

":K" retains a list of all words that match the current pattern modified by the current mask, and the addresses for those words. This list may be SEARCHed again later using the PREVIOUS modifier. All of memory is searched unless RANGE or PREVIOUS is used.

?AX

The search may be controlled asynchronously through the ?AX (Accept) option. The search occurs from the high end of memory toward 0; thus, memory indices printed in response to ?AX WHERE decrease. The different combinations of tokens for the ?AX command follow.

?AX WHERE

The WHERE option asks where the search is.

?AX STOP

The STOP option stops the search and reprompts the user with a ":READY" message.

?AX -TEXT
?AX +TEXT

The +TEXT and -TEXT options turn the listing of the text on and off, respectively.

?AX -KEEP

The -KEEP option suppresses the formation of the list of kept matches.

?AX SKIP

The SKIP option abandons the current range and skips to the next range (if any).

Examples:

```
SEARCH RANGE STK 368 BOSR TO LOSR

SEARCH MOMDESC RANGE 0 TO END :K         %FIND MOMS IN MEMORY

SEARCH REFS 381F6 PREV : T           %FIND MOMS POINTING HERE
```

See also

ASN	79
BOX (IOM SYSTEMS)	80
MASK	122
PATTERN	138

SHAREMEM (B 5900 and B 6900 SYSTEMS)

The SHAREMEM command displays the status of shared memory on a loosely-coupled system. This command is invalid on systems using MCP/AS.

<sharemem>

```
    -- SHAREMEM ---------------|
                |           |
                |-<number>-|
```

Semantics:

SHAREMEM

The status of shared memory on all memory modules is displayed.

SHAREMEM <number>

The status of the memory module identified by <number> is displayed.

SIB

<sib>

```
    -- SIB -- VIA --<simple address>--|
```

Semantics:

The SIB command causes the SIB whose descriptor is located at <simple address> to be printed.

A Structure Information Block (SIB) is the local data for each user of a data base. The SIB consists of a group of workareas and SIRWs for local data set/set manipulation followed by a D[04] portion for each structure invoked by the data base user. A descriptor to the SIB is located in the user stack.

Examples:

The following command causes the SIB whose descriptor is located at the absolute address 13702 to be printed:

```
    SIB VIA 13702
```

The following command causes the SIB whose descriptor is located in stack EE offset 1B from BASE, to be printed:

```
    SIB VIA STK EE BASE #1B
```

STACK

The STACK command causes the contents of a stack, formatted and interpreted, to be displayed.

```
<stack>

    -- STACK --<number>-------------------------|
                        |                      |
                        |-<number>------------|
                                   |          |
                                   |-<number>-|
```

Semantics:

STACK <number>

> The contents of the entire stack identified by <number> are formatted and interpreted. All of this information is displayed, starting with the top of the stack and continuing to the base of the stack.

STACK <number> <number>

> The contents of the stack identified by the first <number> are formatted and interpreted, then displayed. Here only the section of the stack from the starting or upper bound offset into the specified stack (identified by the second <number>) to the base is displayed.

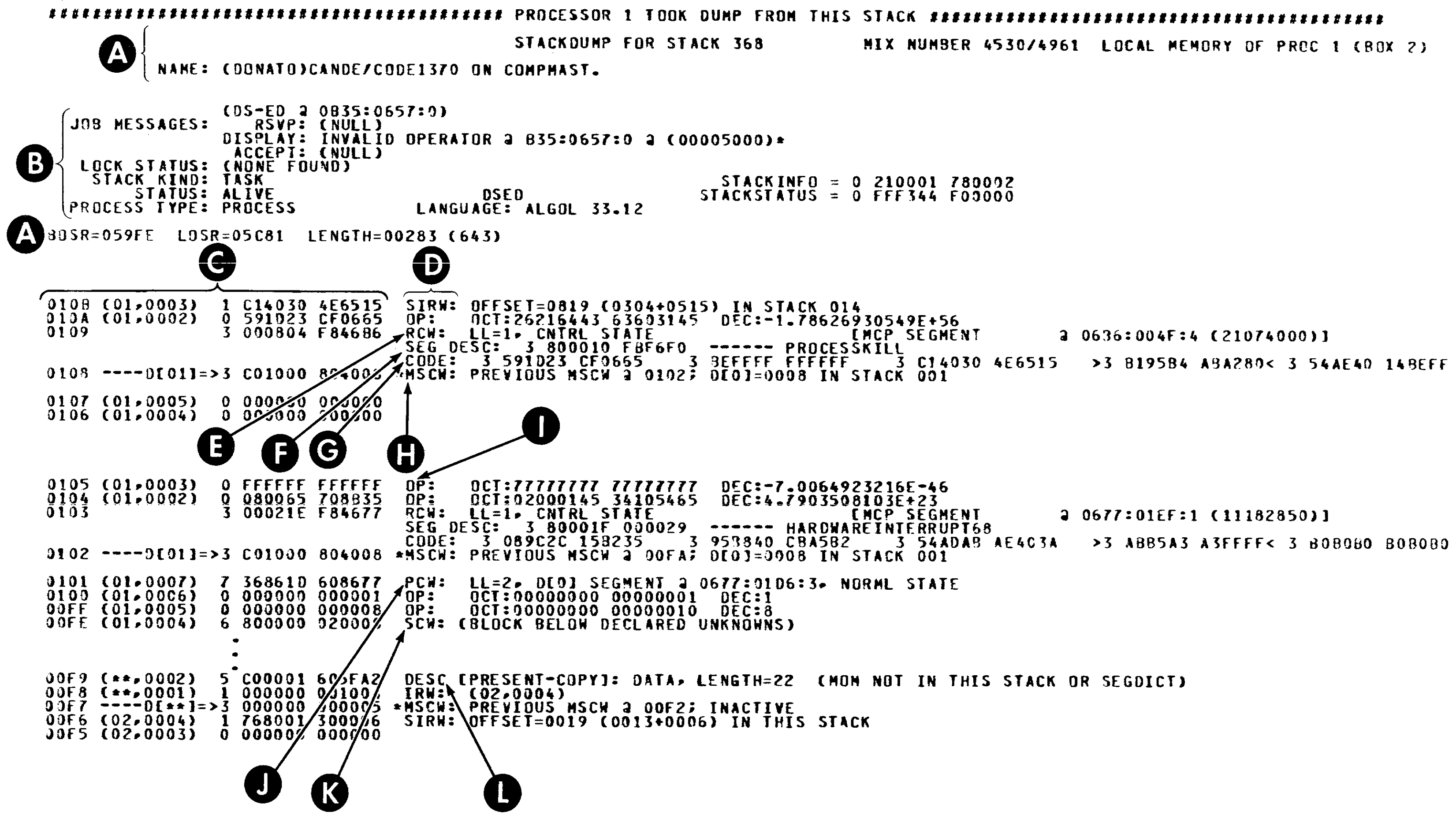
STACK <number> <number> <number>

> As in the preceding case, the contents of the stack starting from a given offset are formatted and interpreted, then displayed. Here an ending or lower-bound within the stack is indicated by the third <number>, further limiting what portion of the stack will be displayed. (Refer to Figures 5-7, 5-8, and 5-9 for sample stack dumps.)

Examples:

The example below lists stack 368 from 10B through F5:

```
    STACK 368  10B   F5
```



```
############################################ PROCESSOR 1 TOOK DUMP FROM THIS STACK ############################################
                                            STACKDUMP FOR STACK 368         MIX NUMBER 4530/4961  LOCAL MEMORY OF PROC 1 (BOX 2)
  NAME: (DONATO)CANDE/CODE1370 ON COMPMAST.

                 (DS-ED @ 0835:0657:0)
 JOB MESSAGES:      RSVP: (NULL)
                 DISPLAY: INVALID OPERATOR @ 835:0657:0 @ (00005000)*
                  ACCEPT: (NULL)
  LOCK STATUS: (NONE FOUND)
   STACK KIND: TASK                                                    STACKINFO = 0 210001 780002
       STATUS: ALIVE                    DSED                         STACKSTATUS = 0 FFF344 F00000
 PROCESS TYPE: PROCESS            LANGUAGE: ALGOL 33.12
BOSR=059FE   LOSR=05C81   LENGTH=00283 (643)

0108 (01,0003)   1 C14030 4E6515  SIRW:  OFFSET=0819 (0304+0515) IN STACK 014
010A (01,0002)   0 591D23 CF0665  OP:    OCT:26216443 63603145  DEC:-1.78626930549E+56
0109             3 000804 F84686  RCW:   LL=1, CNTRL STATE                [MCP SEGMENT        @ 0636:004F:4 (21074000)]
                                  SEG DESC:  3 800010 FBF6F0   ------ PROCESSKILL
                                  CODE:  3 591D23 CF0665     3 BEFFFF FFFFFF     3 C14030 4E6515    >3 8195B4 ABA280< 3 54AE40 14BEFF
0108 ----D[01]=>3 C01000 804008  *MSCW:  PREVIOUS MSCW @ 0102; D[0]=0008 IN STACK 001

0107 (01,0005)   0 000000 000000
0106 (01,0004)   0 000000 000000

0105 (01,0003)   0 FFFFFF FFFFFF  OP:    OCT:77777777 77777777  DEC:-7.0064923216E-46
0104 (01,0002)   0 080065 708835  OP:    OCT:02000145 34105465  DEC:4.7903508103E+23
0103             3 00021E F84677  RCW:   LL=1, CNTRL STATE                [MCP SEGMENT        @ 0677:01EF:1 (11182850)]
                                  SEG DESC:  3 80001F 000029   ------ HARDWAREINTERRUPT68
                                  CODE:  3 089C2C 158235     3 958840 CBA5B2     3 54ADAB AE4C3A    >3 ABB5A3 A3FFFF< 3 B0B0B0 B0B0B0
0102 ----D[01]=>3 C01000 804008  *MSCW:  PREVIOUS MSCW @ 00FA; D[0]=0008 IN STACK 001

0101 (01,0007)   7 36861D 608677  PCW:   LL=2, D[0] SEGMENT @ 0677:01D6:3, NORML STATE
0100 (01,0006)   0 000000 000001  OP:    OCT:00000000 00000001  DEC:1
00FF (01,0005)   0 000000 000008  OP:    OCT:00000000 00000010  DEC:8
00FE (01,0004)   6 800000 020000  SCW:  (BLOCK BELOW DECLARED UNKNOWNS)
                 .
                 .
                 .
00F9 (**,0002)   5 C00001 605FA2  DESC [PRESENT-COPY]: DATA, LENGTH=22  (MOM NOT IN THIS STACK OR SEGDICT)
00F8 (**,0001)   1 000000 001000  IRW:   (02,0004)
00F7 ----D[**]=>3 000000 000005  *MSCW:  PREVIOUS MSCW @ 00F2; INACTIVE
00F6 (02,0004)   1 768001 300006  SIRW:  OFFSET=0019 (0013+0006) IN THIS STACK
00F5 (02,0003)   0 000000 000000
```

Figure 5-7. Stack Analysis (Beginning)

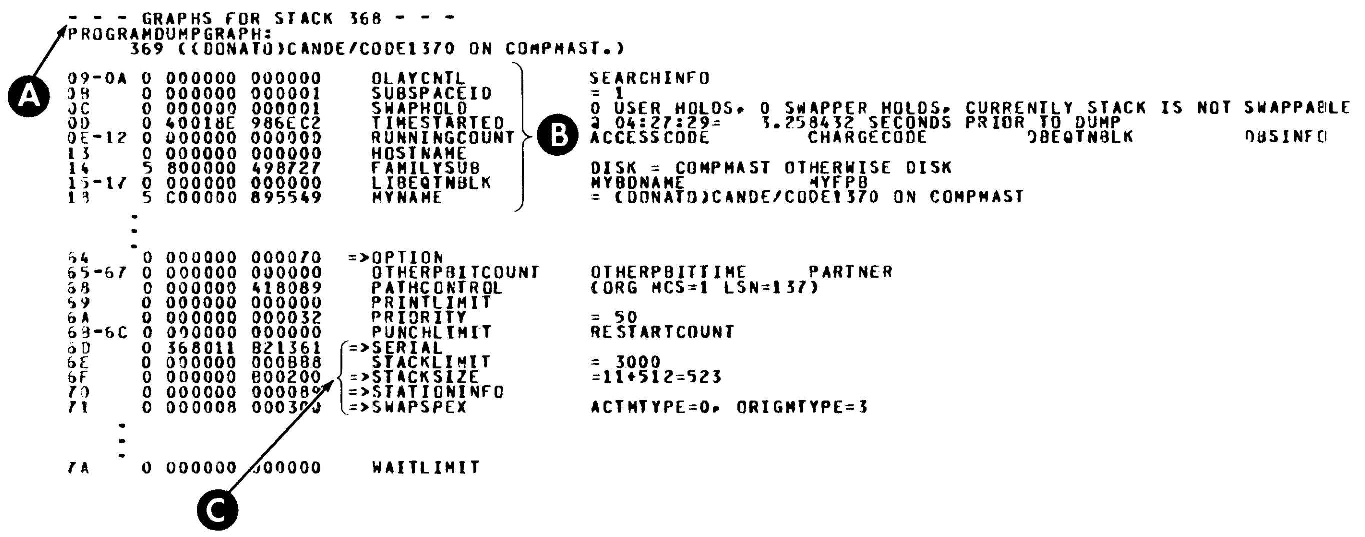
ITEM	DESCRIPTION
A	STACK NUMBER, MIX NUMBER, NAME OF THE STACK, ABSOLUTE MEMORY ADDRESS OF THE BASE OF THE STACK AREA, ABSOLUTE MEMORY ADDRESS OF THE UPPER LIMIT OF THE STACK AREA, AND LENGTH OF THE STACK AREA IN HEXADECIMAL AND DECIMAL NOTATION.
B	INDICATION AS TO WHETHER OR NOT THE STACK WAS DSED, ODT MESSAGES BUILT FOR STACK (RSVP, DISPLAY, OR ACCEPT), PROCESS TYPE, KIND OF STACK, LANGUAGE IN WHICH THE PROGRAM WAS WRITTEN, MARK-LEVEL-PATCH OF THE COMPILER THAT PRODUCED THE PROGRAM, AND STATUS OF THE STACK AT DUMP TIME.
	NOTE THE ACTUAL DUMPING OF THE STACK IS DONE FROM THE S REGISTER DOWN, ONE WORD PER LINE.
C	RELATIVE ADDRESS AND ADDRESS COUPLE (WHEN APPLICABLE) PRECEDE THE HEX PRINTOUT OF EACH WORD.
D	STUFFED INDIRECT REFERENCE WORD (SIRW) PRINTOUT INDICATES THE OFFSET INTO THE SPECIFIED STACK.
E	RETURN CONTROL WORD (RCW) PRINTOUT INCLUDES THE LEXICOGRAPHICAL LEVEL, WHETHER IN CONTROL OR NORMAL STATE, THE SEGMENT NUMBER AND RELATIVE ADDRESS WITHIN THAT SEGMENT WHERE PROCEDURE ENTRY OCCURRED, AND THE SYMBOLIC CODE LINE SEQUENCE NUMBER OF THAT MCP PROCEDURE.
F	SEGMENT DESCRIPTOR IS FOLLOWED BY THE NAME OF THE MCP PROCEDURE REFERENCED BY THE RCW.
G	LEFT AND RIGHT BROKEN BRACKETS IN THE CODE LINE INDICATE THE WORD WHERE PROCEDURE ENTRY OCCURRED.
H	MARK STACK CONTROL WORDS ARE PRECEDED BY AN ASTERISK FOR CLARITY; FOR EACH MSCW, THE POSITION OF THE PREVIOUS MCSW IS INDICATED.
I	OPERANDS ARE PRINTED IN OCTAL, DECIMAL AND EBCDIC ACCORDING TO THE FOLLOWING CONDITIONS: 1. DECIMAL OPERAND VALUES CAN BE FORMATTED IN INTEGER, FIXED-POINT OR EXPONENTIAL NOTATION, AS APPROPRIATE. 2. EBCDIC OPERANDS ARE SO PRINTED ONLY IF ALL CHARACTERS ARE GRAPHICS. HOWEVER, IF AN OPERAND CONTAINS RIGHT-JUSTIFIED EBCDIC CHARACTERS WITH NULL FILL, THE EBCDIC CHARACTERS ARE PRINTED. 3. NO EXPANSION OF ANY KIND IS MADE INTERACTIVELY UNLESS THE APPROPRIATE EXPANSION MODES ARE SET (REFER TO MODE IN THIS SECTION). IF ALL MODES ARE SPECIFIED, BCL IS NOT SET AND MUST BE SET SEPARATELY, IF DESIRED.
J	PCW PRINTOUT INCLUDES THE LEXICOGRAPHICAL LEVEL OF THE PROCEDURE BEING ENTERED, THE SEGMENT NUMBER AND RELATIVE ADDRESS WITHIN THAT SEGMENT WHERE PROCEDURE ENTRY OCCURS, AND THE INTERRUPT STATE.
K	SOFTWARE CONTROL WORD (SCW) PRINTOUT INDICATES TYPE OR DIMENSION OF ARRAYS (OR BOTH), KIND OF DECLARATION (FILE, INTERRUPT, OR FAULT), OR PRESENCE OF A NON-LOCAL GO TO.
L	WORD DATA DESCRIPTOR (DESC), INDICATING DATA PRESENCE IN, OR ABSENCE FROM, MAIN MEMORY, WHETHER DATA-ITEM IS A COPY OR THE ORIGINAL DESCRIPTOR, TYPE AND LENGTH OF DATA-ITEM, AND (IF REQUIRED) ADDRESS OF THE DATA-ITEM.
	DESCRIPTORS FOR ACTIVE PIBS ARE IN A SPIBVECTOR PARALLEL TO THE STACK VECTOR.
	STEP INDEX WORD (SIW) PRINTOUT INDICATES CURRENT VALUE, INCREMENT VALUE, AND FINAL VALUE OF AN ITERATION STATEMENT.
	DOUBLE-PRECISION OPERAND (DPOP) PRINTOUT INDICATES THE VALUE OF EACH WORD IN OCTAL, SCIENTIFIC NOTATION, AND DOUBLE-PRECISION SCIENTIFIC NOTATION.

Figure 5-7. Stack Analysis (Beginning) (Cont.)

```
  0011 ----D[02]=>3 FF954C E08010 *MSCW: PREVIOUS MSCW @ 0001; D[1]=54CE IN STACK 3F9
  0010               5 270000 742ACA  THEFILE
  000F               0 000000 000000  STRINGPOOLMOM
  000E               0 000000 000000  STRINGPOOLMARK
  000D               0 000000 000000  LOCKCOUNT
  000C               0 000000 300019  JUNK
  000B               0 000000 000000  BLOWBY
  000A               5 000000 400000  SEGMNTARRAY
  0009               0 000000 000000  CONDLISTDESC
  0008               0 000000 000000  DATDESC          A
  0007               5 800010 0754DF  AITDESC
  0006               7 368000 0000CA  STACKINFOTOP
  0005               5 800000 6059BC  OLAYINFODESC
  0004               6 800000 000000  OLAYCW
  0003               0 000000 000000  INTMAPLINKS
  0002               0 008000 0020E2  MEMLOC
  0001               3 768000 000000  STUFFITMSCW
  0000               0 000000 000001  TOSCWD
B SPIBVECTOR[368] = 5 C00008 3954CE    MOM @ 06E53, DECLARED BY STACK 082
  00     3 C01000 804000      PIBMSCW
  01     0 000000 00041C      ACTUALBOXMASK
  02     0 000000 000369      CODELINKS          SEG DICT = 369
  03     0 000000 21000C      COMPILERINFO
  04     0 400000 E3841E      COREINTEGRAL       = -14922782
  05     0 000000 000653      COREINUSE          = 1619
  06     0 000000 00054F      COREINUSESAVED     = 1359
  07     0 000000 000000      FORCEDOLAYTIME
  08     0 000000 00000C      GRAPHHEADWORD
```

ITEM	DESCRIPTION
A	STACK ATTRIBUTES.
B	SPIBVECTOR. DESCRIPTORS FOR ACTIVE PIBS ARE IN THE SPIBVECTOR THAT IS PARALLEL TO THE STACK VECTOR.

Figure 5-8. Stack Analysis (End)



```
- - - GRAPHS FOR STACK 368 - - -
PROGRAMDUMPGRAPH:
     369 ((DONATO)CANDE/CODE1370 ON COMPMAST.)

09-0A 0 000000 000000     OLAYCNTL           SEARCHINFO
0B    0 000000 000001     SUBSPACEID         = 1
0C    0 000000 000001     SWAPHOLD           0 USER HOLDS, 0 SWAPPER HOLDS, CURRENTLY STACK IS NOT SWAPPABLE
0D    0 40018E 986EC2     TIMESTARTED        @ 04:27:29=    3.258432 SECONDS PRIOR TO DUMP
0E-12 0 000000 000000     RUNNINGCOUNT       ACCESSCODE        CHARGECODE        DBEQTNBLK          DBSINFO
13    0 000000 000000     HOSTNAME
14    5 800000 498727     FAMILYSUB          DISK = COMPMAST OTHERWISE DISK
15-17 0 000000 000000     LIBEQTNBLK         MYBDNAME          MYFPB
18    5 C00000 895549     MYNAME             = (DONATO)CANDE/CODE1370 ON COMPMAST
      .
      .
      .
64    0 000000 000070   =>OPTION
65-67 0 000000 000000     OTHERPBITCOUNT     OTHERPBITTIME      PARTNER
68    0 000000 418089     PATHCONTROL        (ORG MCS=1 LSN=137)
69    0 000000 000000     PRINTLIMIT
6A    0 000000 000032     PRIORITY           = 50
6B-6C 0 000000 000000     PUNCHLIMIT         RESTARTCOUNT
6D    0 368011 B21361   =>SERIAL
6E    0 000000 000888     STACKLIMIT         = 3000
6F    0 000000 800200   =>STACKSIZE          =11+512=523
70    0 000000 000089   =>STATIONINFO
71    0 000008 000300   =>SWAPSPEX           ACTMTYPE=0, ORIGMTYPE=3
      .
      .
      .
7A    0 000000 000000     WAITLIMIT
```

ITEM	DESCRIPTION
A	GRAPHS.
B	TASK OR PIB ATTRIBUTES.
C	ARROW INDICATES THAT THE ATTRIBUTE IS ACTIVE.

ERROR MESSAGE: ACCEPT: BAD INDEX ARRAY

ONE OF THE THREE ARRAYS FOR TASKS (PIBS), STACKS, AND FIBS DOES NOT HAVE THE PROPER CHECK FLAG. ANY RESPONSE CAUSES THAT ARRAY TO BE MARKED BAD AND PROCESSING TO CONTINUE.

FILE NOT PRESENT/IN ERROR; <file name>

A USER FILE IS NOT PRESENT FOR THE PURPOSE OF READING ITS LINEINFO, OR THE LINEINFO IS INCONSISTENT. PROCESSING CONTINUES. (THIS MESSAGE IS NOT DISPLAYED, BUT IS PRINTED IN THE STACK AT THE FIRST REFERENCE TO THE USER FILE.)

Figure 5-9. Stack Analysis (Graphs)

STOP

The STOP command terminates SYSTEM/DUMPANALYZER. A synonym for STOP is BYE.

```
<stop>

    -- STOP --|
```

SUBPORT

The SUBPORT command prints analysis of a specified subport. Groups of subports are associated with different port files. Each subport of a port file can be connected to a different process. There are two syntaxes for this command; one for BNA Version 1 and one for BNA Version 2. (Refer to Figure 5-6.)

BNA Version 1 Syntax:

```
-- SUBPORT ---<index>------------|
              |                 |
              |- AT --<address>-|
```

Semantics:

SUBPORT <index>

The subport with the given index is analyzed. The index is found in port data structure.

SUBPORT AT <address>

The subport at the given address is analyzed.

BNA Version 2 Syntax:

```
-- SUBPORT ---<index>---------------------<port index>----|
              |         |      | |         |              |
              |         |- OF -| |- PORT -|              |
              |                                          |
              |- AT --<address>--------------------------|
```

Semantics:

```
SUBPORT <index> <port index>
SUBPORT <index> OF <port index>
SUBPORT <index> PORT <port index>
SUBPORT <index> OF PORT <port index>
```

The subport with the given index within the port having the given port index is analyzed. The port index is found in the library information word of the File Information Block (FIB). The subport index specifies the index of the subport within the port.

```
SUBPORT AT <address>
```

The subport at the given address is analyzed.

SUMMARY

The SUMMARY command provides a list of the stacks in the machine and the status of these stacks at the time that a dump was taken. Active stacks and the stack that took the dump are also indicated. (Refer to Figure 5-10.) All swap jobs displayed by SUMMARY can be identified by the pound sign (#) placed after the job type.

```
<summary>

    -- SUMMARY --|
INPUT: SUMMARY
SS STK     MIX #             NAME                KIND  STATUS     TYPE
-- --- --------- ------------------------------ ---- ----------- ----
G  001       0/0 *SYSTEM/MCP32252A/FMLYINX002 0 MCP  FROZEN
1  002       0/0 ETERNALIR/F/1.                 I.R. WAITING     RUN
2  003       0/0 ETERNALIR/L/2.                 I.R. WAITING     RUN
3  004       0/0 ETERNALIR/F/3.                 I.R. WAITING     RUN
G  014       0/0 IDLE/PROCESSOR/1.              I.R. READY QUEU  RUN
G  015       0/0 IDLE/PROCESSOR/2.              I.R. READY QUEU  RUN
G  016       0/0 IDLE/PROCESSOR/3.              I.R. ALIVE       RUN  ACT
G  01A       0/0 ETERNALIR.                     I.R. WAITING     RUN
G  01B       0/0 AREAMANAGER.                   I.R. ASLEEP      RUN
G  01D 3825/3826 CONTROLLOR.                    I.R. WAITING     RUN
G  01E       0/0 ANABOLISM.                     I.R. WAITING     RUN
2  029 3837/4231 *"32.253[12172026]"/CANDE/STAC TASK WAITING     PROC
G  043           *SYSTEM/TSLINTRINSICS ON SYS32 INTR UNEMPLOYED
2  044           *SYSTEM/TSLINTRINSICS ON SYS32 INTR UNEMPLOYED
3  045           *SYSTEM/TSLINTRINSICS ON SYS32 INTR UNEMPLOYED
1  046           *SYSTEM/TSLINTRINSICS ON SYS32 INTR UNEMPLOYED
3  048 3831/3831 "DCP/3".                       I.R. WAITING     RUN
2  049 3832/3832 "DCP/2".                       I.R. WAITING     RUN
2  082 3837/3837 *SYSTEM/CANDE ON PACK.         TASK WAITING     RUN
2  08A           *SYSTEM/CANDE ON PACK.         SEGD UNEMPLOYED
1  08D 3839/3839 *SYSTEM/GENERALSUPPORT/32253A  LIB  FROZEN      RUN
1  08E           *SYSTEM/GENERALSUPPORT/32253A  SEGD UNEMPLOYED
3  09A 4272/4273 (P)OBJECT/UTIL/HELP ON COMPMAS TASK WAITING     PROC
3  09B           (P)OBJECT/UTIL/HELP ON COMPMAS SEGD UNEMPLOYED
2  09C 3842/3842 *SYSTEM/GENERALSUPPORT/32253A  LIB  FROZEN      RUN
2  09F           *SYSTEM/GENERALSUPPORT/32253A  SEGD UNEMPLOYED
2  0A2 3837/3846 *"32.253[12172026]"/CANDE/STAC TASK WAITING     PROC
2  0AB 4705/4705 *SYSTEM/PLISUPPORT ON SYS32.   LIB  FROZEN      RUN
2  0AC           *SYSTEM/PLISUPPORT ON SYS32.   SEGD UNEMPLOYED
G  0AD       0/0 PORTLIB.                       I.R. WAITING     RUN
2  0BD 3859/4713 *OBJECT/ED.                    TASK WAITING     PROC
2  0C5 3837/3876 *"32.253[12172026]"/CANDE/STAC TASK WAITING     PROC
3  0CF 3879/3879 *SYSTEM/GENERALSUPPORT/32253A  LIB  FROZEN      RUN
3  0D0           *SYSTEM/GENERALSUPPORT/32253A  SEGD UNEMPLOYED
1  0DC 3873/3884 *OBJECT/ED ON SYS33.           TASK WAITING     PROC
1  0DD           *OBJECT/ED ON SYS33.           SEGD UNEMPLOYED
G  126 3915/3915 *SYSTEM/GENERALSUPPORT/32253A  LIB  FROZEN      RUN
G  127           *SYSTEM/GENERALSUPPORT/32253A  SEGD UNEMPLOYED
1  13C 4746/4746 "THURS B-UP".                  JOB  TO BE CONT  JOBS
1  13D           FOR "THURS B-UP".              SEGD UNEMPLOYED
1  143 4746/4748 LIBRARY/MAINTENANCE.           I.R. READY QUEU  CALL
2  17B           *OBJECT/ED ON DISK.            SEGD UNEMPLOYED
2  19A 4124/4125 (12748JOSEPH)OBJECT/DMSCLASS/O TASK WAITING     CALL
2  19B           (12748JOSEPH)OBJECT/DMSCLASS/O SEGD UNEMPLOYED
2  19C 4126/4126 (12748JOSEPH)DMSCLASSDB.       DBTS WAITING     RUN
2  19D           (12748JOSEPH)ACCESSROUTINES/DM SEGD UNEMPLOYED
1  1AA 4129/4130 (12749JOSEPH)OBJECT/DMSCLASS/I TASK WAITING     CALL
1  1AB           (12749JOSEPH)OBJECT/DMSCLASS/I SEGD UNEMPLOYED
1  1AC 4131/4131 (12749JOSEPH)DMSCLASSDB.       DBSD UNEMPLOYED
3  288 4760/4865 *OBJECT/ED ON SYS32.           TASK WAITING     PROC
3  289           *OBJECT/ED ON SYS32.           SEGD UNEMPLOYED
3  28D 4866/4866 DAILYUPDATED.                  JOB  WAITING     JOBS
3  28E           FOR DAILYUPDATED.              SEGD UNEMPLOYED
3  294 4871/4871 "DAILY-UPDATED".               JOB  TO BE CONT  JOBS
3  295           FOR "DAILY-UPDATED".           SEGD UNEMPLOYED
2  296 4871/4872 *SYSTEM/FILECOPY.              TASK ALIVE       CALL ACT
2  297           *SYSTEM/FILECOPY ON DISK.      SEGD UNEMPLOYED
3  363 4755/4959 (JONES)OBJECT/UTIL/BEND ON CON TASK WAITING     PROC
3  364           (JONES)OBJECT/UTIL/BEND ON CON SEGD UNEMPLOYED
1  368 4530/4961 (DONATO)CANDE/CODE1370 ON COMP TASK ALIVE       PROC DUMP
1  369           (DONATO)CANDE/CODE1370 ON COMP SEGD UNEMPLOYED
```

ITEM	DESCRIPTION
A	THE MEMORY SUBSYSTEM FOR EACH STACK OF A B 6800 MULTIPROCESSOR SYSTEM: G FOR GLOBAL, 1 FOR PROCESSOR 1 (BOX 2), AND SO FORTH. THE COLUMN IS BLANK FOR A STACK WHOSE BOX IS NOT CURRENTLY ASSIGNED.

Figure 5-10. Summary

SWAPANAL

The SWAPANAL command analyzes SWAPPER internal queues and prints a list of the SWAPPER parameters, if they are available. This command is invalid on systems using MCP/AS.

```
<swapanal>

    -- SWAPANAL --|
```

Example:

The following is an example of the SWAPPER parameter analysis produced by the SWAPANAL command:

```
  INPUT: SWAPANAL


  **********SWAPPER PARAMETER ANALYSIS**********
  *** RUNNING *** SWAPCOREMAX = 90 SLOTS(89100 WORDS)
  ACTUALCORESIZE = 90 SLOTS(89100 WORDS) MINTIMESLICE = 3 SECONDS
  MAXSLICENR = 7 RATIO = 4
  MAXCORESLOTS = 30SLOTS(29700 WORDS) EXPRESERVE  = 0 SLOTS(0 WORDS)
  EXPMAXCORE = 0 SLOTS(0 WORDS) PRIORITYBIAS = 0
  UTILIZATIONBIAS = 6 EXPRESSMAXTIME = 1 SECOND(S)
  IOBIAS = 0 MEMORYBIAS = 0
  MAXIOSIZE = 15 SLOTS(14850 WORDS) MINCHUNKSIZE = 90 SLOTS(89100 WORDS)
  NOSWAPTRANSTATE

  ******** PACKS ********
  SWAPPACK(SWHDR=259) SWAPPER2(SWHDR=260) SWAPPACK1(SWHDR=261)

  ********************SWAPPER QUEUE ANALYSIS*****************
  *****************TASKS WAITING FOR CORE****************
  STACK 27A, PRIO = (007,000)    -   (*BS/OBJECT/BS9150 ON SYSTEMPACK.)
  STACK 308, PRIO = (007,000)    -   (*BC/OBJECT/BC0150 ON SYSTEMPACK.)
  STACK 180, PRIO = (007,000)    -   (*BC/OBJECT/BC0100 ON SYSTEMPACK.)
  ************NO INACTIVE TASKS IN CORE****************
```

Under the heading SWAPPER PARAMETER ANALYSIS are all the SWAPPER parameters set by the system operator. Under the heading PACKS is a list of all the packs used by SWAPPER.

TRACE

The TRACE command causes the current tracetable, or a portion of the tracetable, to be printed. The tracetable is generated by the trace facility, a general purpose MCP debugging aid available when the MCP compile-time option TRACE is set. The TRACE facility allows for tracetable entries to be made at predefined points in the MCP called CONDITIONALDUMP stops, each of which is associated with a particular statement in the MCP. Each stop has two parameters, a stop number and an info word. The stop number is a two digit hex number that is different for each stop. A new set of CONDITIONALDUMP stops is in effect for each mark level release. Information about the conditions under which the stops are reached is given in the tracetable entry. The tracetable holds about 1650 entries in a circular queue; when the last slot has been filled, the next entry overwrites the first entry.

The TRACE facility is controlled by the primitive ODT commands ??CD and ??TRACE. (Refer to the Operator Display Terminal (ODT) Reference Manual for information about the ??CD and ??TRACE commands.)

<trace>

```
              |<----------------------------|
              |                             |
  -- TRACE ---------------------------------|
                |                         |
                |-/1\-----------<number>--|
                |          |              |
                |-/1\- STACK -|           |
                |          |              |
                |-/1\- CPM ---|           |
                |                         |
                |          |<---------|   |
                |          |          |   |
                |-/1\- CD ---<number>-----|
```

TRACE

Entering "TRACE" alone prints the entire tracetable. "TRACE" may be followed and modified by <number>, STACK <number>, CPM <number> or CD <number>, or a combination of these items.

TRACE <number>

The most recent <number> of entries to the tracetable are printed. If CPM, STACK, or CD are specified, and a <number> precedes them, the preceding <number> indicates how many trace entries are to be searched (not printed).

TRACE <number> STACK <number>

The most recent <number> of entries to the tracetable are searched, and those in the specified STACK <number> are printed.

TRACE <number> CPM <number>

The most recent <number> of entries to the tracetable are searched, and and those in CPM <number> are printed.

TRACE <number> CD <number>

The most recent <number> of entries to the tracetable are searched, and those with CD (conditional dump) <number> are printed.

Example:

The following is the output from the command "TRACE 11" run on a B 6900 dump. Ordinarily, all the headings would appear on one line; of necessity they are split up into two groups below.

```
CD#    P2 PARAMETER       IN PROCEDURE           @ SEQ# / RCW    CPM
---    -- ------------    -- ---------------     - ----------    ---
18     0 000039 000100    HARDWAREINTERRUPT68     (11307600)      4
18     5 0B1669 080100    HARDWAREINTERRUPT68     (11307600)      4
18     3 02C879 000100    HARDWAREINTERRUPT68     (11307600)      4
08     0 000000 00000B    UPDATESTACKPROCS        (12782000)      4
18     0 000008 100091    HARDWAREINTERRUPT68     (11307600)      4
08     0 000000 000014    UPDATESTACKPROCS        (12782000)      4
18     0 000009 000800    HARDWAREINTERRUPT68     (11307600)      4
08     0 000000 00000B    UPDATESTACKPROCS        (12782000)      4
07     1 000000 00039E    FORKHANDLER             (15944000)      4
08     0 000000 000009    UPDATESTACKPROCS        (12782000)      4
32     0 000000 000029    INITIATE                (18619000)      4


STK    CALLED BY PROCEDURE     @ SDI:PIR:PSR / SEQUENCE
---    ------ -- ---------     - --- --- ---   --------
00B    MESSER                   0FE3:005A:2    (77617120)
00B    MESSER                   0FE3:0063:5    (77616934)
00B    PROCESSKILL              0B87:0000:0    (21035610)
00B    WAITP                    0AA7:0199:2    (12471200)
014    DISABLEP                 0AA8:02FA:0    (12866500)
014    HARDWAREINTERRUPT68      0B41:0048:1    (11323200)
00B    CUTBACKAITARRAY          0B08:0000:0    (04973000)
00B    HARDWAREINTERRUPT68      0B41:019A:1    (11371000)
00B    ONESECONDBURDEN          0AA8:0195:3    (12374700)
009    MULTIPLEWAIT             0AA7:0279:3    (12542400)
022    ANABOLISM                0B6D:005E:4    (17275000)
```

USE

The USE command lists commands that were saved by the KEEP command. (Refer to the KEEP command.)

<use>

```
    -- USE ---<number>----|
          |           |
          |- ? ------|
```

Semantics:

USE <number>

The command that was saved and labeled <number> by the KEEP command is listed. <number> must evaluate to a decimal integer between 0 and 9.

USE?

A list of the commands saved by the KEEP command is printed.

Example:

```
INPUT: USE ?

 00  : HDR
 01  : MD 1C33 FOR 6
 02  : MD STK 6A BOSR FOR 3
```

See also

KEEP. 112

WHERE

The WHERE command displays the location of a specified <MCP global name>. The location is expressed as the offset relative to D[0].

All aliases for MCP stack cells are retained and recognized by the WHERE command. When WHERE is entered from the ODT, the system response is displayed on that ODT.

```
<where>

    -- WHERE --<MCP global name>--|
```

Example:

```
INPUT : WHERE FROCK
 268 FROCK
```

WHO

The WHO command displays the MCP global name for the specified D[0] offset (cell). An outer-block procedure PCW is shown as SEG:PIR:PSR from the D[0] stack image. For an outer-block procedure code segment, the names, PCW D[0] offsets, and PCWs are shown for all procedures in that segment.

All aliases for MCP stack cells are retained and are returned by the WHO command. When WHO is entered from the ODT, the system response is displayed on that ODT.

```
<who>

    -- WHO --<number>--|
```

Examples:

```
 INPUT: WHO 0316
 316 MAXQUEUERS

 INPUT: WHO 320
 320 (PCW 7D6:0044:0) CCCHECK FORMORE

 INPUT: WHO 321
 321 (SEG)
   :0000:0 (PCW   035) WAITIO
   :003D:0 (PCW   043) DISKWAIT
```

5.4 ERROR MESSAGES

The following error messages are displayed if an abnormal condition occurs.

ACCEPT: WRONG CODE FILE -- OK OR RESTART.

The code file timestamp on disk did not match the timestamp on the dump tape. The operator should enter OK to use the code file on disk or RESTART to cause the code file to be closed and SYSTEM/DUMPANALYZER to be suspended. DUMPANALYZER proceeds if OK is specified.

DISPLAY: DUMP TAPE HAS BAD INFORMATION IN RECORD <#>, AT LOCATION <#>.

Data on the dump tape failed the consistency check, causing DUMPANALYZER to terminate.

DISPLAY: BAD DATA RECOVERY IN RECORD <#>, AT LOCATION <#>.

Data on the dump tape failed the consistency check, but redundancies in the way that these data were stored on the tape allowed limited recovery of that data. DUMPANALYZER then proceeds.

DISPLAY: ERROR UNABLE TO GENERATE GLOBAL IDENTIFIERS, CAUSE = <cause>.

One of four <cause>s caused failure in the MCP global identifier routine. Two of these <cause>s indicate either improper compilation, a DUMPANALYZER bug, or code file corruption (LEVEL and SIZE). I/O exceptions give rise to the other two <cause>s, RDMCP (code file) and RDFILE (internal file). Global name generation is terminated, NONAMES is invoked, and DUMPANALYZER analysis proceeds. DUMPANALYZER can be rerun with the DEBUG command to get a program dump of the terminated global identifier routine.

DISPLAY: CANNOT ANALYZE - MCP INCOMPATIBLE WITH DUMPANALYZER.

The MCP level recorded on the dump tape did not match the level of DUMPANALYZER. This incompatibility can be avoided by ensuring that the level of DUMPANALYZER is always exactly the same as the level of the MCP that wrote the dump tape.

DISPLAY: CANNOT ANALYZE -- USE PREVIOUS DUMP ANALYZER

The level of the MCP as recorded on the dump tape was lower than the level compiled into SYSTEM/DUMPANALYZER, causing DUMPANALYZER termination. The proper level of SYSTEM/DUMPANALYZER must be used for a rerun.

DISPLAY: BAD DUMPANALYZER INPUT CARDS

DUMPANALYZER was unable to decipher an input card. The card image appears on the printout with a line of asterisks (*) pointing to the unknown word, and processing terminates.

DISPLAY: BAD MCP STACK POINTER

DUMPANALYZER found that the stack vector descriptor at D[0]+2 did not address present memory. This condition is usually due to a premature end-of-file condition on the dump tape or to improperly taking the last memory module off-line when no MEMDUMP disk is available, causing termination of processing. DUMPANALYZER should be rerun with RAWDUMP and DEBUG set; DUMPANALYZER then produces a raw dump of the contents of the tape without checking D[0]+2.

DUMPANALYZER FAULT <#> messages are given for the first, second, third, fourth, and tenth faults and for every tenth fault thereafter (20, 30, 40, and so forth).

6 INDEXED SEQUENTIAL ACCESS METHOD

This section describes a set of software routines that implement indexed sequential access methods of storage and retrieval of data records. Indexed sequential access method, hereafter referred to as ISAM, allows a keyed file to be processed in both random and serial fashion. This section is intended for use as a reference document for experienced programmers.

This section contains some subsections that apply to both standard and primitive ISAM and other subsections that apply only to primitive ISAM. A discussion of the differences between standard and primitive ISAM is given directly following this introduction. The first four subsections, ending with "Practical Considerations", apply to both standard and primitive ISAM. The last three subsections deal only with primitive ISAM. Most of the material concerning standard ISAM, including lists of exception codes, is contained in the PL/I Language Reference Manual in the chapters titled "Data Description" and "Input/Output", and in the COBOL Language Reference Manual in the section describing ANSI74 enhancements.

The ISAM facility is only accessible using the COBOL (with or without the $ANSI74 option), PL/I, and ALGOL compilers. The COBOL74 and RPG compilers do not use ISAM; they use KEYEDIO. (Refer to KEYEDIO for information about KEYEDIO). ISAM-like capabilities are also available through DMSII. For further information, refer to the DMSII DASDL Reference Manual and the DMSII User Language Interface Software Operation Guide.

The support procedures for ISAM are contained in the PLISUPPORT library. All documentation notes pertaining to ISAM appear under the heading PLISUPPORT. To initiate PLISUPPORT, the SL (System Library) ODT command is used. (Refer to the Operator Display Terminal (ODT) Reference Manual.)

See also
    KEYEDIO 217

6.1 PROGRAM INTERFACE - PRIMITIVE AND STANDARD ISAM

The following paragraphs define the two ways of invoking ISAM, the primitive method and the standard method.

PRIMITIVE ISAM

A set of ISAM procedures for creating and updating ISAM files are listed later in the "General Implementation Information For Primitive ISAM" and "ISAM Procedures" sections. Primitive ISAM involves direct calls of the procedures that perform the ISAM functions rather than use of special language syntax. Parameters must be passed explicitly. Each procedure returns a value indicating its results. In primitive ISAM, the program must detect exception conditions by interpreting the result word whose contents are explained at the end of this section. ISAM exception conditions do not cause program termination when primitive ISAM is used. Primitive ISAM allows the highest amount of selection and control. ALGOL must use primitive ISAM. COBOL may select either primitive or standard ISAM. PL/I must use standard ISAM.

STANDARD ISAM

The standard interface simplifies programming effort by allowing normal, higher level language input/output statements such as READ and WRITE to be used, rather than directly invoking the ISAM procedures. No loss of efficiency results when standard ISAM is used. PL/I must use the standard method. COBOL may select either the standard or primitive method. PL/I and COBOL each have unique implementation features.

The standard interface includes the use of ISAM file options. Functionally, ISAM file options are similar to file attributes but exist only for ISAM files. Unlike file attributes, ISAM file options may not be assigned a value or be modified by control cards or programmatic file attribute statements. Each ISAM file option is assigned a value when the file is created and opened as OUTPUT. Examples of COBOL ISAM file options include the settings for KEYSPERENTRY, AREAOVERFLOW, and FILEOVERFLOW. Some keyed file options for PL/I include KEYLENGTH, KEYORDER, FILEOVERFLOW, and WAITUPDATEIO.

Standard ISAM deals with exception conditions differently than primitive ISAM. ISAM procedures return an information word to reflect the results of the ISAM invocation. In standard ISAM, the compiler emits code for observing the results and initiating appropriate action. ISAM exception conditions may occasionally cause program termination when standard ISAM is used. Tables listing the exception codes for standard ISAM are given in the COBOL Language Manual and the PL/I Language Reference Manual.

6.2 STRUCTURE OF ISAM FILES

An ISAM file consists of three logical sections within one physical file:

a. The prime data area

b. The prime data area overflow space

c. The file overflow area

These three sections are defined in the following paragraphs.

PRIME DATA AREA

The prime data area holds all the keyed data records that are entered when the file is first created. The maximum size of this area (in records) can be determined by multiplying the values of two file attributes: AREAS and AREASIZE. The number of prime data area rows is specified by the value of the AREAS attribute. The number of records in each prime data area row is specified by the value of the AREASIZE attribute. File space is automatically reserved by the ISAM program for nondata purposes (coarse and fine tables); the amount reserved is determined by the values of the attributes AREAS and AREASIZE.

ISAM files do not assume the default values of a normal file for AREAS and AREASIZE because these values should be carefully chosen for optimum performance. The values associated with AREAS and AREASIZE must be specified when creating (and opening as OUTPUT) an ISAM file, but they do not need to be specified at any other time.

When an ISAM file is created, all unused space contained in the final row of the prime data area is incorporated into overflow space for the final row, and totally unused prime data area rows are incorporated into the file overflow area.

DATA OVERFLOW AREA

The data overflow area of the file is the unoccupied data area. Records added after file creation are always placed in an overflow area. Two types of physical overflow areas are provided: the prime data area overflow space and the file overflow area.

Prime Data Area Overflow Space

Area overflow space may be provided in each row containing prime data and is specified when the file is created (it is opened as OUTPUT at file creation time.) In standard ISAM, row overflow space is indicated by a file option such as AREAOVERFLOW in COBOL. In primitive ISAM, row overflow space is indicated through a parameter to the ISOPEN procedure. When records are deleted, the occupied space can be returned to the overflow pool in the prime data area row in which the record resides. The deleted record option, set to ON when the file is created (and opened as OUTPUT), specifies the disposition of the space occupied by deleted records.

File Overflow Area

A file overflow area outside of the prime data area may also be specified when the file is created (and opened as OUTPUT). Again, a file option in standard ISAM or the ISOPEN procedure in primitive ISAM indicates the size of the overflow area. Records are placed in the file overflow area only after all available overflow space in the specific prime data area row in which the record would normally reside has been filled.

TABLES FOR LOCATING DATA

Two levels of tables are used by the ISAM procedures: fine tables and coarse tables. Each prime data area row of the file contains a fine table. The fine table is a list of keys and associated file addresses. One key (and address) is placed in the table for each n records, where n is a program-selected value when the ISAM file is first created. The fine table is stored at the physical end of its corresponding file area.

The entire file has one coarse table that contains pairs of keys and addresses. Each key entry is identical to the first key entry of the corresponding fine table, and the address entry is the address of the fine table rather than the address of a data record. The coarse table is stored at the physical beginning of the file overflow area. Therefore, an ISAM file always has at least one physical row of file overflow space.

DATA RECORD LINKS

ISAM data records are linked together in a logical sequence. Each record automatically contains both a forward and backward link to its logical successor and predecessor. A link is an address of a data record. The first data record contains a backward link that is zero, and the last data record contains a forward link that is zero. Forward links are used to locate data records. Both forward and backward links are used to insert and delete data records. Data record links are the innermost level of file structure in an ISAM file.

The coarse table serves to locate a fine table; the fine table, in turn, locates a data record. The data record links are utilized in following the trail to the desired record when necessary. Data records are not physically moved to accommodate additions and deletions. Instead, the data record links are modified, so that file changes are handled in a logical rather than physical fashion. Because links are physically contained in every data record, ISOPEN must increase the record size to provide space for the links. Increasing record size is accomplished by rounding the original record size up to the nearest full word and then adding one more word to contain the links.

ISAM'S MANAGEMENT OF OVERFLOW AREAS

When an ISAM file is created, unoccupied space may be reserved in each prime data area row, and at least one entire row may be reserved for overflow records. The fine table that corresponds to a row also contains information that provides a link to the next available unoccupied space. Overflow space that is reserved when an ISAM file is created is allocated in serial fashion. If deleted record space is subsequently made available for reuse (an option selected by the program), the deleted record(s) are linked into the available record chain for the corresponding area and reassigned on a last-in, first-out (LIFO) basis. Record space made available for reassignment is reused before unused space is assigned.

The coarse table contains the link to the next available space in the file overflow area. Space assignment in the file overflow area is the same as overflow assignment in a prime data row. New records are not placed in the file overflow area if they can be placed in the prime data area. A given record is never eligible for placement in more than one prime data area row, and the only alternative placement for it is in the file overflow area.

6.3 PLANNING FOR ISAM FILES

ISAM provides a specific set of capabilities that must be considered during preparation for application programs and systems. Trade-offs can be made to favor a particular course of action. All features of the ISAM procedures are not available to every mode of operation and every language. The following paragraphs discuss the factors that should be considered when planning for ISAM files.

MAXIMUM NUMBER OF RECORDS

The maximum number of records that can be contained in a single ISAM file is 16,777,215. Some of this space is required for a coarse table, fine tables, and an INFO record. Data records can occupy the remaining space.

COARSE TABLE SIZE

One coarse table is created for the entire file.

Coarse table size is determined by the number of prime data area rows used during file creation; however, the number of rows used cannot exceed the value of AREAS requested because the coarse table cannot expand. Not more than 999 prime data area rows may be requested because at least one row is required for file overflow. The default value of AREAS is 1. One entry is made in the coarse table for each prime data area row. The table is contracted when fewer prime data area rows are used than specified. Key length also has a direct effect on table size.

The coarse table size must not exceed 393,210 bytes, or an error message will be issued. To compute the number of bytes needed for coarse table size, refer to the following subsection.

Computing Coarse Table Size

All units are bytes (8-bit characters).

Coarse table size =

```
        (number of table entries * (key length + 3)
             + 24 + BLOCKSIZE - 1) DIV BLOCKSIZE * BLOCKSIZE.
```

Record space loss to coarse table =

```
        coarse table size DIV BLOCKSIZE * number of records per block.
```

FINE TABLE SIZE

One fine table is created for each prime data area row. A prime data area is a row of the ISAM file that was written into while the file was being created. All other rows of the file are allocated to file overflow and do not contain fine tables. In any ISAM file, all fine tables have identical size.

A fine table ratio is specified to determine the number of entries in each fine table. The ratio may range between 1 and 63; the default value is 1. When duplicate records are permitted, only the first record in the duplicate set is eligible for entry in the fine table or is counted in meeting the fine table ratio. Key length also directly affects fine table size. Space for the fine table is automatically allocated for the fine table by the ISAM program.

The fine table size must not exceed 393,210 bytes, or an error message is given. To compute the number of bytes needed for fine table size, refer to the following subsection.

Computing Fine Table Size

All units are bytes (8-bit characters).

Fine table size =

```
          (number of table entries * (key length + 3)
                + 24 + BLOCKSIZE - 1) DIV BLOCKSIZE * BLOCKSIZE.
```

Record space loss to fine table =

```
          fine table size DIV BLOCKSIZE * number of records per block
                * number of fine tables.
```

INFO RECORD SIZE

The first record of each ISAM file is a special record that contains attribute information about that particular ISAM file. This INFO record is essential for proper access of the ISAM file's data records. The current length of the INFO record is 7 words. One data record space is normally required to contain the INFO record, but more may be used if the data record length is less than 7 words (42 bytes).

AREAS AND AREASIZE

The file attributes AREAS and AREASIZE are more important to ISAM files than to non-ISAM files. Both attributes must be specified when creating a new ISAM file, but are not needed at other times. The default value is 1 for both attributes (non-ISAM file defaults are different). The value of AREAS indicates the number of prime data area rows expected for the ISAM file; the value of AREAS cannot exceed 999. When an ISAM file is first created, a few more areas than needed should be specified. The value of AREASIZE, as specified by the program, indicates the number of data records per prime data area row.

The value of the AREASIZE attribute is automatically increased by the ISAM program to allow the fine table to be written in the same area (or row) of the file as the data it represents. The value of the AREASIZE attribute is also increased by the number of overflow records per area that are specified by a file option in standard ISAM or by an ISOPEN procedure in primitive ISAM. Similarly, the value of the AREAS attribute is increased by the number of file overflow areas specified by a file option in standard ISAM or by an ISOPEN procedure in primitive ISAM. This increase is very similar to allowing the file to expand via the FLEXIBLE attribute; the increase does not affect the size of the coarse table. When a given ISAM file is closed, the AREAS and AREASIZE attributes are reset to their original values.

MINIMUM RECORD SIZE (MINRECSIZE)

ISAM does not provide for variable length records. Therefore, the value of the MINRECSIZE attribute should be 0 or identical to the value of the MAXRECSIZE attribute.

MAXIMUM RECORD SIZE (MAXRECSIZE)

The value chosen for the MAXRECSIZE attribute is entirely dependent upon the needs of the program and the absolute limits allowed by the system. ISAM increases the value of the MAXRECSIZE attribute by at least 1 word (6 bytes) and at most 11 bytes. The program should not assign a new value to the MAXRECSIZE of a given ISAM file except when the file is first created and opened. When the ISAM file is closed, the MAXRECSIZE attribute is reset to its original value. The maximum usable values for MAXRECSIZE are 65,534 words or 65,535 characters.

BLOCKSIZE

The BLOCKSIZE attribute is used in association with the MAXRECSIZE attribute to determine the number of records per block. The value of BLOCKSIZE is always changed by ISAM. When the program specifies a nonzero value for BLOCKSIZE, ISAM retains the number of records per block specified by the program. The value of BLOCKSIZE is increased to accommodate larger records. When the program specifies a BLOCKSIZE of zero, ISAM computes a new value for BLOCKSIZE in order to conserve disk storage space. After the value MAXRECSIZE has been increased, ISAM computes the smallest number of records that exactly fits into a multiple of 30-word disk segments. The BLOCKSIZE attribute is restored to its original value when the file is closed.

EXCLUSIVE USE

The EXCLUSIVE attribute is a Boolean attribute that is set to TRUE by ISAM when an ISAM file is opened as INPUT-OUTPUT. ISAM files may not be shared except when all programs open the file as INPUT only.

FINE TABLE RATIO

The programmer may select any value from 1 through 63 for the fine table ratio; a value of 1 signifies a table entry for each record, and 63 signifies a table entry for each 63 records. The most successful choice is highly data- and program-dependent. A small value is generally appropriate when records are often accessed randomly rather than sequentially. In order to improve or restore performance, programmatic reorganization of the file may be desirable after a number of changes have occurred.

KEY LENGTH

Some key modes allow specific maximum key lengths of 5, 6, or 11 bytes (Refer to the "Mode of key" section under the ISOPEN procedure for a full list of key modes.) Character keys of 4-bit or 8-bit characters are limited only by the 14-bit field that contains key length. The maximum usable key lengths are 8-bit characters for 1020 bytes and 4-bit characters for 508 bytes (1016 hex characters). Shorter keys yield faster performance and smaller fine and coarse tables.

See also
    ISOPEN. 202

KEY OFFSET

A 15-bit field is allowed that permits an offset of 32767 for 8-bit characters and 16382 for 4-bit characters.

PRACTICAL CONSIDERATIONS

In an unstable environment, ISAM files may become corrupted, a condition in which the coarse table, fine tables, or data record links do not concur. Corruption of ISAM files may occur when the physical file on disk has not yet been updated to reflect the changes that have been made to buffers in memory. If an event occurs that prevents the updated buffers from being written into the physical file, the file may become corrupted. Use of standard ISAM helps to prevent this situation.

In those cases where the program terminates prematurely (because of an invalid index, a divide-by-zero error, a DS, or some other reason), standard ISAM performs an orderly close of the ISAM file, while primitive ISAM may not be able to close the file properly. However, the possibility of file corruption exists only after the file has been opened as INPUT-OUTPUT and writes or deletions have occurred. File damage is by no means inevitable, and two file options (WAITUPDATEIO and PHYSICALUPDATE) are available to further reduce such a possibility. (Refer to the WAITUPDATEIO and PHYSICALUPDATE options discussed under the ISOPEN procedure.)

ISAM files may not be specified as input or output files to the SORT utility, except in PL/I. They must be read and written by INPUT or OUTPUT procedures. Other system software may also encounter similar situations when attempting to process ISAM files in a direct fashion without use of the ISAM procedures.

ISAM files may be accessed simultaneously by several programs, if they all open the file as INPUT. Only one program may access the file while it is open as OUTPUT or INPUT-OUTPUT.

Direct I/O is used by ISAM procedures to access the data. Therefore, the ISAM file must be a direct file. In primitive ISAM, the program must declare the file as direct. The compiler properly declares the file in the standard method. The direct arrays used by the ISAM procedures are created by ISOPEN and returned by ISCLOSE. The program does not need other direct arrays to access the data.

ISAM files may not be used in an Inter-Program Communication (IPC) environment in which the file is passed from one task to another.

6.4 IMPLEMENTATION FOR PRIMITIVE ISAM PROCEDURES

ISAM is implemented by a set of procedures in the PLISUPPORT support library. Symbolics for these procedures are contained in the PLISUPPORT symbol file. The procedures called directly from programs are as follows:

1. ISOPEN - Open and set up file.
2. ISCLOSE - Close file.
3. ISREAD - Randomly read a record.
4. ISWRITE - Add a record to the file.
5. ISREADNEXT - Read the next sequential record.
6. ISREWRITE - Rewrite the record just read.
7. ISKEYWRITE - Randomly rewrite a record.
8. ISDELETE - Delete a record.

These ISAM procedures must be used to OPEN, CLOSE, create, and access ISAM files. Non-ISAM files cannot be accessed by these procedures. Non-ISAM file OPEN, CLOSE, READ, and WRITE statements and accesses via file attributes are not disallowed; however, the use of any of them may cause unexpected results that may be detrimental to the integrity of the ISAM file. The ISCLOSE procedure should be used to CLOSE the ISAM file. An implicit CLOSE of an ISAM file caused by exiting a block does not properly save the file.

A $SET INSTALLATION 1 record must appear at the beginning of the symbolic for all ALGOL programs that invoke ISAM procedures.

6.4.1 ISAM PROCEDURES

The following subsections describe the system procedures that collectively institute the ISAM methodology. Each procedure is defined in terms of its function or functions within the general ISAM operating method and in terms of its interaction, if any, with other ISAM procedures. The standard program interface does not require direct use of these procedures and their parameters. However, the primitive program interface uses these procedures directly.

ISOPEN

The ISOPEN procedure opens an ISAM file for INPUT, OUTPUT, or INPUT-OUTPUT. ISAM files require additional information not provided for non-ISAM files. The ISOPEN procedure uses and creates the additional information according to the method of file opening. Non-ISAM files may not be opened by this procedure.

ISAM CALLING SEQUENCE:

```
ALGOL:  RS := ISOPEN(FILE,VALUE,STACK);
```

```
COBOL (PRIMITIVE): COMPUTE RS = ISOPEN (FILE, VALUE, STACK).
```

RS	The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
FILE	The ISAM file being opened. FILE must be declared as a DIRECT FILE in ALGOL and COBOL.
VALUE	Specifies how the ISAM file is to be opened: 1 - open as INPUT 2 - open as OUTPUT 3 - open as INPUT-OUTPUT

INPUT and INPUT-OUTPUT require an existing ISAM file. OUTPUT always means creation of a new file.

OUTPUT requires specification of additional file information. High order bits in this parameter (VALUE) are utilized to convey certain information used for file creation. Bits and fields contained in this parameter are as follows:

47:1	Separate key (PL/I only).
46:15	Offset of the key, in bytes, from the start of the record. It is the TRUE (zero-relative) offset. A value of zero means the start of the record.
31:2	Open action (open as INPUT or as INPUT-OUTPUT only). 0 - Open the file. 1 - Use PRESENT attribute to open. 2 - Use AVAILABLE attribute. 3 - Not used.
29:14	Actual key length in bytes.
15:4	Mode of key. Values are as follows: 0 - BINARY (6-byte maximum) 1 - 8-Bit character 2 - 8-Bit unsigned numeric (11 bytes maximum) 3 - 8-Bit MSD signed numeric (11 bytes maximum) 4 - 8-Bit LSD signed numeric (11 bytes maximum) 5 - 4-Bit characters 6 - 4-Bit unsigned numeric (5 bytes maximum) 7 - 4-Bit MSD signed numeric (6 bytes maximum) 8 - 4-Bit LSD signed numeric (6 bytes maximum)
11:1	Duplicate key option. If this field is zero, records with duplicate keys may not be added to the file. If this field is one, duplicates are chained in first-in, first-out (FIFO) sequence. A duplicate key condition exists when the keys in two records are equal.

10:1	Deleted record option. If this field is zero, deleted records are physically delinked and their record space becomes available for reuse. If this field is one, deleted records are flagged by having 4"FF" (all bits ON) placed in the first byte of the record. Records marked as deleted can be retrieved using READNEXT if bit 2 of this parameter word equals 1.
9:1	Sequence option. If this field is zero, the file is in ascending sequence. If this field is one, the file is in descending sequence.
8:6	Fine table ratio. During file creation, this field controls the number of entries made in the fine table(s) and specifies the number of unique records to be added to the file between fine table entries.
2:1	See deleted record option. If this field is zero, deleted records are not visible to the program. If this field is one, deleted records may be visible to the program if the deleted record option is set to 1 and READNEXT is used.
1:2	Open type (previously described). 0 - invalid 1 - INPUT 2 - OUTPUT 3 - INPUT-OUTPUT

STACK specifies the first of four consecutive words in the program's stack. The location of the first word is used by ISOPEN to build data descriptors in all four words. The location is retained in the FIB for use as long as the file remains open. The program must provide the space by declaring the four consecutive stack locations preferably with four type REAL variables in ALGOL and four usage COMP-1 in COBOL. The four words are not usable by the program while the ISAM file is open. A program reference to any of the four words during the time the file is open causes an invalid operator.

Additional file information is conveyed to ISOPEN by use of the first of the four consecutive words.

47:24	Number of overflow records per prime record area row. This field is used only when the file is opened output. At file creation, this field is used to increase the value of AREASIZE specified for the file. The new, larger value of AREASIZE becomes a permanent attribute of the file. Unoccupied space, large enough to contain the number of records specified by this field, is allocated in each row of the file.
23:1	Wait update I/O field. If one, this field causes ISAM procedures to wait for I/O completion of all outstanding I/Os before returning to the program. This field cannot be set to one if the ANSI74 option is set in COBOL.
22:1	Physical update I/O field. If one, this field causes the ISAM procedures to initiate I/Os for all buffers and tables that have been modified and need to be rewritten. This field cannot be set to one if the ANSI74 option is set in COBOL.
21:6	Unused at present but reserved for future implementation.
15:16	Number of file overflow area rows. This field is used only when the file is opened as OUTPUT. At file creation, this field is used to increase the value of the AREAS attribute specified for the file. The new, larger area becomes a permanent attribute. Any areas represented by this field are not used to contain prime data. Prime data area rows unused at file creation are, however, placed in the file overflow area pool. Therefore, when in doubt, it is better to make the AREAS attribute larger.

ISCLOSE

The ISCLOSE procedure closes an ISAM file in an orderly fashion. The normal CLOSE statement is not sufficient to close an ISAM file properly. Certain additional file information is saved within the file by the ISCLOSE procedure, and the four consecutive stack words are cleared or restored. Non-ISAM files may not be closed by this procedure.

PROGRAM CALLING SEQUENCE:

ALGOL: RS := ISCLOSE (FILE, TYPE);

COBOL (PRIMITIVE): COMPUTE RS = ISCLOSE (FILE, TYPE).

RS	The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
FILE	The ISAM file to be closed.
TYPE	A numeric value that specifies how the ISAM file is to be closed:

0	Close the file and release it from the program. This numeric value indicates a normal close. The file does not remain on disk unless it has been previously locked.
1	Close the file with lock. The file is entered into the directory and remains on disk. Any previous file with a duplicate name may be removed.
2	Close the file and purge its entry from the directory. Any disk space occupied by the file becomes available for reassignment by the system.

ISREAD

The ISREAD procedure reads a record in a random fashion using the program-supplied key. If the program-supplied key matches a record in the ISAM file, the matching record is returned. When no matching record exists, the next logically sequential record is returned.

The ISAM file must be opened as INPUT or INPUT-OUTPUT in order to read records. The ISREAD procedure may not be used to read non-ISAM files. The ISAM file must be opened by the ISOPEN procedure before the ISREAD procedure can be used to read records.

PROGRAM CALLING SEQUENCE:

```
ALGOL:  RS := ISREAD (FILE, KEY, AREA);

COBOL (PRIMITIVE): COMPUTE RS = ISREAD (FILE, KEY, AREA).
```

RS	The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
FILE	The ISAM file.
KEY	The key identifying the record to be read. For ALGOL, KEY must a pointer. For COBOL, KEY must be a data item.
AREA	The area to contain the record to be read. The AREA must be at least as large as the record. For ALGOL, AREA must be a pointer. For COBOL, AREA must be a data item.

ISWRITE

The ISWRITE procedure writes a record, using the provided key, from the provided area. This procedure never overwrites or rewrites previously existing records but always adds (or attempts to add) records to the file.

When the ISAM file is opened as OUTPUT, a new file is created. The ISWRITE procedure is used to create coarse and fine tables in addition to placing records into the ISAM file. Records must be presented in the sequence specified by the program when the ISAM file is created. Duplicate record acceptance depends on the setting of the duplicate key option.

When the ISAM file is opened as INPUT-OUTPUT, a previously existing file is utilized. Records need not be presented in any special sequence. The records are written into area overflow or file overflow space and appropriately linked into the ISAM file.

The file must be opened as OUTPUT or INPUT-OUTPUT. The ISWRITE procedure may not be used for non-ISAM files. The file must be opened by the ISOPEN procedure before the ISWRITE procedure can be used.

PROGRAM CALLING SEQUENCE:

```
ALGOL:  RS := ISWRITE (FILE, KEY, AREA);

COBOL (PRIMITIVE):  COMPUTE RS = ISWRITE (FILE, KEY, AREA).
```

RS	The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
FILE	The ISAM file.
KEY	The key identifying the record. The value associated with KEY must match the value in the key location in the record. For ALGOL, KEY must be a pointer. For COBOL, KEY must be a data item.
AREA	The record to be written. The area must be at least as large as the record. For ALGOL, AREA must be a pointer. For COBOL, AREA must be a data item.

ISREADNEXT

The ISREADNEXT procedure reads the next logically sequential record. The record returned to the program is the record whose key immediately follows in sequence after the most recent record obtained by ISREAD or ISREADNEXT.

The ISAM file must be opened as INPUT or as INPUT-OUTPUT by the ISOPEN procedure before the ISREADNEXT procedure can be used. procedure. Non-ISAM files may not be accessed by the ISREADNEXT procedure.

The purpose of the ISREADNEXT procedure is to provide a sequential processing capability. When ISREADNEXT is used in combination with ISREAD and ISREWRITE, records may be sequentially processed and updated for all or part of any ISAM file. ISREADNEXT may be used to read an entire ISAM file in a sequential manner.

PROGRAM CALLING SEQUENCE:

```
ALGOL:  RS := ISREADNEXT(FILE, AREA);

COBOL (PRIMITIVE): COMPUTE RS = ISREADNEXT (FILE, AREA).
```

RS	The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
FILE	The ISAM file.
AREA	The area to contain the record to be read. The area must be at least as large as the record. For ALGOL, AREA must be a pointer. For COBOL, AREA must be a data item.

ISREWRITE

The ISREWRITE procedure replaces the record previously read with the data currently in the record area. This procedure allows records to be updated. Either the ISREAD or ISREADNEXT procedure must immediately precede the ISREWRITE procedure. The key contained in the record to be rewritten must match the key in the record that was read by the immediately preceding file operation.

The file must be an ISAM file and must be opened as INPUT-OUTPUT. Additional records may not be added to the file by the ISREWRITE procedure.

PROGRAM CALLING SEQUENCE:

```
ALGOL:  RS := ISREWRITE(FILE, AREA);

COBOL (PRIMITIVE): COMPUTE RS = ISREWRITE (FILE, AREA).
```

RS	The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
FILE	The ISAM file.
AREA	The record to be written. The area must be at least as large as the record. For ALGOL, AREA must be a pointer. For COBOL, AREA must be a data item.

ISKEYWRITE

The ISKEYWRITE procedure provides a random access update capability for ISAM files. It replaces a currently existing record from the file with the record provided by the program.

The file must be an ISAM file and must be opened as INPUT-OUTPUT. The ISKEYWRITE procedure provides update capability and does not add additional records to the file.

PROGRAM CALLING SEQUENCE:

```
ALGOL:  RS := ISKEYWRITE(FILE, KEY, AREA);

COBOL (PRIMITIVE): COMPUTE RS = ISKEYWRITE (FILE, KEY, AREA).
```

RS	The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
FILE	The ISAM file.
KEY	The key identifying the record to be replaced. The value associated with KEY must match the value in the key location in the record passed in the AREA parameter. For ALGOL, KEY must be a pointer. For COBOL, KEY must be a data item.
AREA	The record to be written. The area must be at least as large as the record. For ALGOL, AREA must be a pointer. For COBOL, AREA must be a data item.

ISDELETE

The ISDELETE procedure drops or deletes records from the file. When duplicate records are allowed, the first (that is oldest) record is deleted. This procedure provides random access delete capability.

The file must be an ISAM file and must be opened as INPUT-OUTPUT. The ISDELETE procedure deletes the first (that is, oldest) record with a matching key. The record may be physically or logically deleted depending on the DELETED RECORD OPTION.

PROGRAM CALLING SEQUENCE:

```
ALGOL:  RS := ISDELETE(FILE, KEY);

COBOL (PRIMITIVE): COMPUTE RS = ISDELETE (FILE, KEY).
```

RS	The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
FILE	The ISAM file.
KEY	The key identifying the record to be deleted. For ALGOL, KEY must be a pointer. For COBOL, KEY must be a data item.

6.5 ISAM I/O RESULT INFORMATION

The ISAM procedures return result values to the calling program. These result values indicate success or failure of the program request. Each value returned is a 48-bit word. In primitive ISAM, the result word is type BOOLEAN in ALGOL and COMP-1 or COMP in COBOL. In standard ISAM, PL/I uses CONDITION CODES and COBOL uses FILE STATUS. The PL/I Language Reference Manual describes CONDITION CODES. FILE STATUS is described in the COBOL Language Manual.

PRIMITIVE ISAM

The value returned when primitive ISAM is used is a 48-bit word that is nonzero when an exception condition exists and zero when no exception condition occurs. Specific, individual bits are utilized to indicate the exception condition. If several different exceptions occur, the corresponding bit is turned ON for each condition and creates the possibility of reporting back several exceptions for a single request. The rightmost and least significant bit (bit 0) is used for a specific purpose. Bit 0 is turned ON when any exception condition occurs and turned OFF when no exception exists. The remaining bits convey the following meanings:

1:1	A hardware error, (for example, a parity error), occurred while processing the request. Another bit (7, 8, 9, or 10) is turned ON to further define the problem.
2:1	An attempt was made to read or write beyond end-of-file.
3:1	No record was found in the file whose key matches the requested key.
4:1	No space is available in the file to contain the record just written. (This condition applies for adding records to the file; it does not apply to file creation.)
5:1	A request was made to add a record to the file, and the key contained in the record matched a record that existed in the file. Refer to bit 6.
6:1	A record was added to the file (the key of the record matched an existing record of the file). The duplicate key option permits or disallows this situation. When duplicates are allowed, both bit 5 and bit 6 are ON to indicate that a duplicate record has been added. Refer to bit 5.
7:1	A hardware error occurred in reading a data record. Bit 1 is also ON.
8:1	A hardware error occurred in writing a data record. Bit 1 is also ON.
9:1	A hardware error occurred in reading an ISAM table. Bit 1 is also ON.
10:1	A hardware error occurred in writing an ISAM table. Bit 1 is also ON.
11:1	This bit is not used.

12:1	An attempt was made to open the ISAM file, and the parameters passed to ISOPEN failed to meet one or more requirements. The "first of four stack words" parameter must be a Stuffed Indirect Reference Word (SIRW). The file must be declared in a block that will be entered no sooner than the block in which the four stack words reside. The file must not reside in a different stack from the program doing the open. The block containing the four stack words must also contain a file, an array, or something that causes a tag-6 word for the block. The tags of all four stack words must be zero. The key must be defined to be contained in the records, have a size greater than zero, and have a valid mode.
13:1	An attempt was made to open a non-ISAM file.
14:1	The file has not been opened or the open type does not permit the request. (For example, an ISWRITE on a file opened as INPUT.)
15:1	An ISREWRITE was requested, and the key of the record being rewritten does not match the key in the last record read, or the previous request was not an ISREAD or an ISREADNEXT.
16:1	The ISAM file is being created, and the record just written did not maintain proper file sequence. Records must be presented in sequence during file creation. A duplicate record also causes this bit to be ON when duplicates are not allowed.
17:1	The value of AREAS specified is not large enough to contain the data records written in the prime data area during file creation.
18:1	ISOPEN is requested to open an already open file.
19:1	In an IPC environment, one program closed an ISAM file, and another attempted an I/O after the file was closed.
20:1	An ISWRITE is requested, and the key supplied does not match the key contained in the supplied record.
21:1	The ISAM file is not a direct file.
22:1	An attempt was made to write a record containing the deleted record indicator (hex FF in first byte).
23:1	This bit indicates a PL/I program error condition. The program is requesting an I/O that is not allowed for keyed files. An ON condition is raised in the PL/I program.

24:1	This bit is ON if ISOPEN is requested (by way of an open action) to open the ISAM file using the PRESENT or AVAILABLE file attributes; this bit also indicates that the desired file could not be located. Refer to bits 43:8.
43:8	This field contains the result of testing the PRESENT or AVAILABLE file attributes in the ISOPEN procedure. If the file could not be opened, bit 24:1 is also ON.
46:1	This bit is ON if the physical update I/O action is ON, the wait update I/O option is OFF, and an I/O error occurred as the result of doing an update I/O in the previous invocation of an ISAM procedure. Bit 1:1 is ON, and 8:1 or bit 10:1 is ON.

7 KEYEDIO

The COBOL74 and RPG indexed sequential access method allows a file sequenced by keys to be processed in both random and sequential modes. The COBOL74 and RPG indexed sequential access method is provided by the SYSTEM/KEYEDIO library (KEYEDIO) and has the same capabilities as the ISAM intrinsic in the SYSTEM/PLISUPPORT library. However, files created by PLISUPPORT cannot be referenced by KEYEDIO, nor can files created by KEYEDIO be referenced by PLISUPPORT. Access to KEYEDIO procedures can only be obtained through compiler-generated code.

7.1 PHYSICAL STRUCTURE OF KEYEDIO FILES

KEYEDIO files consist of three logical areas contained within one physical file: coarse tables, fine tables, and data. The size of these areas increases during the life of a file as records are added and deleted. The areas are not distinct from one another and are intermixed throughout the file. Each block of data in a KEYEDIO file contains information for a single type of area. Control information describing the block and how to access it is appended to the beginning of each block in the file.

COARSE TABLES

Coarse tables contain key values that describe fine tables or other coarse tables. One entry exists in the coarse tables for each fine table. Several coarse tables are created if more fine tables are present than can be indexed by a single coarse table. These coarse tables are then ordered by another level of coarse tables. This hierarchy continues until only one table remains at the top level; this table is called the "root table". A coarse table entry consists of a key value equal to the largest key in the next lower table and a pointer to that table.

FINE TABLES

Fine table blocks contain one key entry for each record. A key entry consists of a key value and a pointer to the data record associated with that key value.

DATA AREA

Data blocks contain the user's records. The records are stored in these blocks exactly as they were written, but the record size is increased if the user specifies that relative keys are to be used. This addition is internal to the file only and need not be used when calculating record size (MAXRECSIZE). A data block is accessed by going through fine tables.

LOCATING DATA

To find a specific record, the user's key value is first compared against the key values in the coarse tables. The first key value in the coarse table that is greater than or equal to the user's key value is used to locate the coarse table at the next level. This process continues until a fine table is encountered. The fine table is then searched for the user's key value, and the data referenced by that key entry is returned.

Because fine tables are sequentially ordered, they are linked together so that only the fine tables and the data areas need to be read when accessing the file sequentially.

A set of coarse and fine tables is created for each key defined by the user. Also, coarse and fine tables are created for the relative key.

There is a limit of 48 keys per KEYEDIO data file. If the number of keys exceeds 48, one of the following error messages is given:

```
FILE CONTAINS TOO MANY KEYS (GREATER THAN 48) FOR KEYEDIO TO HANDLE

USER OPEN REQUEST DECLARES TOO MANY KEYS (GREATER THAN 48) FOR
KEYEDIO TO HANDLE
```

7.2 FILE AND KEYEDIO LIBRARY MANAGEMENT

All KEYEDIO file-management routines are contained in the SYSTEM/KEYEDIO library. Information about all of the users of the file is also contained in the library stack. The library mechanism allows multiple users to access the same file and also allows linkage of all users of the same file to the same library stack. The user's position in the file is maintained by a "current record pointer" that points to the fine table entry corresponding to the user's current record in the file.

At file-open time, if the FILEORGANIZATION file attribute is equal to INDEXED or INDEXEDNOTRESTRICTED, the user program is linked to the SYSTEM/KEYEDIO library routines instead of the normal FIBSTACK procedures. The KEYEDIO open routine performs certain checks to ensure file integrity. (Refer to the section titled "Special Topics" in the I/O Subsystem Manual.)

If the file is declared to be of type INDEXED, all keys declared when the file was created must be declared by the user program each time the file is opened, and these keys must match exactly. If the file is of type INDEXEDNOTRESTRICTED, keys not known (not declared) by the user program are still updated.

When the file is updated, recovery information is stored so that file integrity may be maintained in the event of abnormal termination. Because of the overhead involved, this information is not saved when the file is being created.

KEYEDIO files may be used by more than one user program at a time. Any number of user programs may be reading the file, but when one of the programs attempts to update (add, delete, or rewrite) the file, other users of the file are locked out for the duration of the update transaction. This lockout feature allows more than one program to have the file open in update mode, while also preserving the integrity of the data. No mechanism exists for locking out other users for more than a single transaction; record level lockout is not provided.

The KEYEDIO library keeps track of the number of programs currently reading and writing to the file. To do an update, both the reader and writer counts must equal zero. To do a read, the write count must equal zero.

REMOVING AND INSTALLING A KEYEDIO LIBRARY

The KEYEDIO library is written and compiled in NEWP. The base library is frozen permanently. Because the base library is frozen, the following steps must be taken if the SYSTEM/KEYEDIO library needs to be removed from the active library list:

1. Find the mix number of the KEYEDIO library by entering the LIBS (Library Task Entries) Operator Display Terminal (ODT) command. (Refer to the Operator Display Terminal (ODT) Reference Manual for further information about this and other ODT commands.)

2. Enter "<mix number> THAW" at the ODT to remove the KEYEDIO library from the list.

NOTE

KEYEDIO will terminate only when no tasks are using it.

The KEYEDIO library job summary will then be printed.

After the old KEYEDIO library has terminated, a new KEYEDIO library can be installed by using the SL (System Library) ODT command.

THE IMPORTANCE OF NOT DSING THE KEYEDIO LIBRARY

It is imperative that the KEYEDIO library stack and the library stacks it processes for each open KEYEDIO file not be DSed. If these libraries are DSed, KEYEDIO files will be hung or a fatal system dump will eventually occur. In order to prevent these libraries from being DSed accidentally, they are automatically locked by an implicit LP (Lock Program) ODT command when they are initiated.

7.3 PROGRAM INTERFACE

Only COBOL74 and RPG can create files that define keys within a record. All languages may use the INDEXEDNOTRESTRICTED value of the FILEORGANIZATION attribute to access or create files with relative keys. Files created and later opened as INDEXEDNOTRESTRICTED have all keys updated during an update, even if the user has not specified all the keys.

The procedures provided by the KEYEDIO library may only be used through normal compiler I/O constructs. No direct interface exists between the user program and the library.

7.4 INDEXED (KEYEDIO) FILE ATTRIBUTES

There are two kinds of attributes discussed in this section: file attributes that must be set in a special way when indexed files are created and accessed (FILEORGANIZATION, BUFFERS and BLOCKSIZE), and attributes that are internal to SYSTEM/KEYEDIO and can only be accessed by the compilers (ISAMKEYS and ACCESSMODE).

To ensure maximum processing efficiency and minimum use of save memory, the BLOCKSIZE and BUFFERS attributes should be set carefully. (Refer to the I/O Subsystem Reference Manual for descriptions of these and other file attributes.)

Increasing the number of buffers used by KEYEDIO reduces the time needed to process an indexed file at the expense of increased usage of save memory. In contrast, reducing the number of buffers increases processing time and decreases save memory usage.

Increasing the block size to allow for one- or two-level access to data in the indexed file decreases processing time but also increases usage of save memory. Decreasing block size to allow for three- or four-level data access increases processing time and decreases save memory usage.

The needs of a particular installation must be considered when choosing values for the BLOCKSIZE and BUFFERS attributes. Suggestions for how to choose BUFFERS and BLOCKSIZE attributes are given in the following paragraphs.

THE FILEORGANIZATION ATTRIBUTE

The FILEORGANIZATION attribute can be set in one of two ways for an indexed file: indexed files that have relative keys have FILEORGANIZATION attribute equal to INDEXEDNOTRESTRICTED. (RPG files have relative keys by default.) Indexed files that do not have relative keys have FILEORGANIZATION attribute equal to INDEXED. The FILEORGANIZATION attribute must be set each time the indexed file is opened. (Refer to the I/O Subsystem Reference Manual for further information about the FILEORGANIZATION attribute.)

SETTING THE VALUE OF THE BUFFERS ATTRIBUTE

The number of buffers used by the KEYEDIO library in processing an indexed file may be controlled by the user.

A program may indicate the number of buffers KEYEDIO is to use by setting the value of the BUFFERS attribute of the indexed file. The value of the BUFFERS attribute is used by the library to determine how many buffers to allocate for processing that indexed file.

Impact of Number of Buffers on Processor Time

Increasing the number of buffers used by KEYEDIO reduces the time needed to process an indexed file at the expense of increased usage of save memory. Reducing the number of buffers decreases save memory usage but increases processing time. In general, programs doing random accesses to the indexed file are more sensitive to the number of buffers than programs doing serial accesses.

The effect of changing the number of buffers on the time needed to process a file tends to be proportional to the reciprocal of the number of buffers; that is, reducing the number of buffers lengthens the processing time more than increasing the number of buffers by the same amount decreases the time. For this reason, decisions to decrease the number of buffers below KEYEDIO's default of ten buffers should be the result of careful consideration and measurement.

Impact of Number of Buffers on Save Memory

Increasing the number of buffers increases the save memory usage of the KEYEDIO library; save memory is equal to the number of buffers multiplied by the actual block size (discussed under "CHOOSING A VALUE FOR THE BLOCKSIZE ATTRIBUTE"). The overall performance of the system, as well as the processing time of the specific application, should be taken into account when deciding to increase the number of buffers.

The value specified for the BUFFERS attribute at file creation time is especially important because this value becomes the permanent default number of buffers to be allocated whenever the file is used later. The time needed to create the file is usually not as strongly affected by the number of buffers as later uses of the file are. For this reason, specifying the proper number for the permanent default is generally of greatest importance when specifying the BUFFERS value to use for file creation.

<u>Rules for Determining the Number of Buffers Used</u>

KEYEDIO determines how many buffers to use for processing an indexed file according to the following rules:

1. First, the value of the BUFFERS attribute specified when the file is created is stored permanently in the file itself. This value is used by the KEYEDIO library both when creating the file and as the default number of buffers to allocate each time the file is subsequently opened.

   If the value of the BUFFERS attribute is not specified at file creation time or the value of the BUFFERS attribute specified is two or less, KEYEDIO uses its default value of ten buffers both for file creation and as the default number of buffers for subsequent opens of the file when the file is being accessed by a single user.

2. Second, the value of the BUFFERS attribute specified when an existing indexed file is opened is used to indicate the number of buffers the KEYEDIO library should use in processing that file. KEYEDIO determines the number of buffers to use for an existing file in the following way:

   a. When the file is not being used by any other programs at the time it is opened, and if the number of buffers specified is 3 or greater, allocate that number of buffers. Otherwise, allocate the default number of buffers established at the time the file was created. Also, allocate an additional buffer for the root table of each key.

   b. When the file is being used by other programs at the time it is opened, increase the total number of buffers allocated by the number of buffers specified or by a default determined by the compiler (as described under c.). When there are multiple users of an indexed file, the buffers are allocated in a common pool by the KEYEDIO library and are shared by all the users.

   c. Some compilers automatically set the value of BUFFERS even when the program has not specified a value for BUFFERS. In particular, the COBOL74 compiler sets BUFFERS to two if it is not specified; the RPG compiler sets BUFFERS to one if it is not specified for an indexed file. In order to keep the number of KEYEDIO buffers at the default value when an existing indexed file is opened, the BUFFERS specification for the indexed file should be set to zero.

The compiler-set defaults will not interfere with the default assignment of 10 buffers at file creation time for an indexed file (because the compiler default is two or less) or the assignment of 10 buffers by default when a single access is made to an already existing indexed file. The compiler default would have an impact when many programs access an indexed file. Each new program accessing the indexed file would increment the number of buffers in use by the compiler default setting.

The maximum value that may be set for the BUFFERS file attribute is 63. The maximum total number of buffers used by the KEYEDIO library is 255. Once this limit is reached, additional buffers will not be allocated, regardless of the BUFFERS specifications of later users.

If the BUFFERS attribute of an indexed file is interrogated, the value returned is the value currently established for that file by the program, not the total number of BUFFERS that are actually being used by KEYEDIO at that time.

CHOOSING A VALUE FOR THE BLOCKSIZE ATTRIBUTE

The actual block size used by KEYEDIO is different from the block size provided by the user because space must be added to round the record size up to an exact multiple of six characters, to allow room for the relative keys of an INDEXNOTRESTRICTED file, and to provide for the ten words of header information in each block. Actual block size is used to calculate how much save memory is actually occupied by the KEYEDIO file. If the actual block size that has been calculated is not satisfactory to the programmer, it may be necessary to adjust the user-specified block size, that is, the attribute BLOCKSIZE.

The user-specified block size is saved in the KEYEDIO file and is returned as the value of the BLOCKSIZE file attribute when the indexed file is open. If this attribute is interrogated when the file is closed, it always returns the value of 30, which is the value that the KEYEDIO library uses when creating the file. (This is a side effect of the fact that the KEYEDIO library manipulates the file using DIRECT I/O.)

The Effect of Block Size on Processor Time

The proper specification of block size is extremely important to the performance of applications that use indexed files because the actual block size (calculated according to the algorithm given under "Calculating Actual Block Size") is used not only for storing the data but also as the size of the key index tables used to access the data. The size of these tables and the number of records in the file determine how many tables must be searched in order to find a particular record. Each additional table that must be searched increases the processor and I/O time that is required to access a record.

The most efficient access is obtained when only a single table must be searched in order to find the key. A single-table search requires that the block size be large enough to hold the keys for all the records in the file; thus, this block size is usually not a practical choice except for files with a small number of records.

The next most efficient access is obtained when only two tables (a coarse table and a fine table) must be searched to find the key. A two-table search requires a block size large enough to hold a number of keys equal to the square root of the number of records in the file. A block size of this value is generally the most suitable choice for all but very small or very large indexed files. A block size smaller than this square root value requires multiple table accesses and noticeably increases the time required for random accesses to the file.

The Effect of Block Size on Save Memory

The buffers used by KEYEDIO occupy save memory. The amount of save memory to be used for a given indexed file can be approximated by multiplying the actual block size (calculated according to the algorithm given in "Calculating Actual Block Size") by the number of buffers to be used for the file.

If the save memory requirements for block sizes that provide one- or two-level access to data are too great, a new block size should be calculated that provides three- or four-level access. This block size may be calculated using the algorithm given under "Calculating User-specified Block Size (2 Level Search)"; but at step 2 compute the cube root or 4th root of the number of records instead of the square root.

Calculating Actual Block Size

The actual block size used by KEYEDIO is different from the BLOCKSIZE attribute specified by the user because space must be added to round the record size up to an exact multiple of six characters, to allow room for the relative keys of an INDEXEDNOTRESTRICTED file, and to provide for the ten words of header information in each block.

The algorithm used by KEYEDIO to compute the actual block size for a file is as follows:

1. Divide the user-specified BLOCKSIZE by the user-specified record size (MAXRECSIZE), truncating any remainder. This gives the user-specified records per block.

2. Round the user-specified record size up to the next multiple of six characters, if it is not already an exact multiple of six characters. Convert this record size to the number of words required to hold the record by dividing by six.

3. If this is an INDEXEDNOTRESTRICTED file, add one (word) to the record size to allow space for the relative key.

4. Compute a trial block size by multiplying the record size in words (calculated in steps 2 and 3) by the user-specified records per block (calculated in step 1). Then add ten words to provide space for the header information in each block.

5. Calculate the actual block size by rounding the trial block size from step 4) up to the next multiple of 30 words, if it is not already an exact multiple of 30 words.

Once the actual block size has been calculated, as many records as will fit are placed in each block. That is, if the rounding process of step 5 adds enough space to the block for additional records, that space will be used, and the actual records per block will be greater than the user-specified records per block calculated in step 1.

Calculating User-Specified Block Size (2 Level)

To calculate the proper block size for an indexed file, assuming the two-level table search is desired, make the following calculation:

1. Calculate the number of records the file will contain over its lifetime.

2. Compute the square root of the number of records. Then multiply this value by an "adjustment" factor to allow for the fact that not all the tables will be completely filled. The result of this computation is the desired number of keys per block.

   The value of the "adjustment" factor is determined by the way the file is created and updated. If the file is created sequentially with the entries for all the keys in ascending order, and few records will be added later, a small "adjustment" factor of 1.1 can be used. If the file is created sequentially, but more records are to be added, use an "adjustment" factor of about 1.3 (or greater, if many records will be added). If the file is created with the entries for some of the keys occurring in random order, use an "adjustment" factor of 2.0.

3. Compute the size of the largest key entry by performing the following steps:

   a. Find the size of the largest key in the record.

   b. Round this size up to the next multiple of six characters, if it is not already a multiple of six characters.

   c. Add six characters to allow space for the key entry's pointer to the data record.

4. Compute the desired block size by multiplying the desired number of keys per block (from step 2) by the size of the largest key entry (from step 3).

5. Round this desired block size up to the next multiple of the record size, if it is not already a multiple of the record size. This last step ensures that the block size chosen is suitable for storing the data records as well as the keys.

The block size calculated by this procedure provides two-level access, but its impact on the system must be determined before deciding that this block size is the correct block size to use. In particular, the effects of the block size on memory usage must be considered. (Refer to "The Effect of Block Size on Save Memory".)

See also
    The Effect of Block Size on Save Memory 230

ATTRIBUTES INTERNAL TO SYSTEM/KEYEDIO

The opening of keyed files requires additional information for which no provision is made in non-keyed files. The attributes ISAMKEYS and ACCESSMODE provide this information. ISAMKEYS and ACCESSMODE may only be accessed by the compilers.

The ISAMKEYS attribute describes the keys declared by the user program. Each key is one word of information put into the file description and marked as an ISAMKEYS attribute (attribute number = 148). The ISAMKEYS attribute is passed to the KEYEDIO library at file-open time. The meaning of the various fields within the value returned by the ISAMKEYS attribute are as follows:

Name	Field	Value	Meaning
KEYFLAGF	[46:01]		Relative or keyed key.
		0	Relative key.
		1	KEYEDIO key.
ALTERNATEKEYF	[45:01]		Alternate or primary
		0	Primary key.
		1	Alternate key.
DUPLICATEF	[44:01]		Duplicates.
		0	No duplicates.
		1	Duplicates.
KEYORGANIZATIONF	[43:01]		Key organization.
		0	Descending.
		1	Ascending.
KEYSIGNPOSITIONF	[39:04]		Sign information.
		0	No sign: alphanumeric data.
		1	Leading separate: numeric data, leading separate sign.
		2	Trailing zone: numeric data, trailing zone.
		3	Leading zone: numeric data, leading zone sign.
		4	Trailing separate: numeric data, trailing separate sign.
		5	Operand.
		6	Two's complement.

KEYEDIO

Name	Field	Value	Meaning
KEYTYPEF	[35:04]		Type of key.
		0	Word.
		2	HEX field.
		4	HEX or EBCDIC field.
		8	EBCDIC.
		8	ASCII.
KEYLENGTHF	[31:16]		Length in KEYTYPEF units.
KEYOFFSETF	[15:16]		Offset in record in KEYTYPEF units.

The ACCESSMODE attribute determines the way the file is accessed. The ACCESSMODE attribute is also contained in the file description. The values and mnemonics of the ACCESSMODE attribute are as follows:

Value	Mnemonic	Meaning
0	SEQUENTIALACCESS	File access is sequential only.
1	RANDOMACCESS	File access is random only.
2	DYNAMICACCESS	File access is sequential and random.

7.5 KEYEDIO PROCEDURES

The procedures that are exported from the KEYEDIO library are described in the following subsections. These procedures are invoked by commands in the appropriate languages; refer to the Report Program Generator (RPG) Reference Manual and the COBOL ANSI-74 Reference Manual.

ISAMOPEN

ISAMOPEN opens a file that has its FILEORGANIZATION attribute equal to INDEXED or INDEXEDNOTRESTRICTED. If a new file is being created, the file is initialized and the key information is saved. Initialization includes creation of a key root table for each key defined by the user program. If the user program is accessing an existing file, the user's file declaration and key information are checked against those of the existing file, and the user's current record pointer and current key of reference are established.

The ISAMOPEN procedure uses the following parameters and returns an open result:

```
ISAMOPEN (ISAMFILE,FILEINFO,OPENTYPE)
```

ISAMFILE	The user's file. When creating a new file, the attributes contained in ISAMFILE are used to set up the keyed file.
FILEINFO	An array that contains information about the program opening the file and the keys declared within that program. The meanings of the values returned from the array are given in the following table. Not all the values provided for are used by the compilers or supported by KEYEDIO.

Field	Value	Meaning
[47:04]		Format level. (Current level is 3.)
	1	Initial implementation.
	2	Language field is valid.
	3	Attribute information is passed.
[43:04]		Status of file.
	0	Closed.
	1	Open Input.
	2	Open Output.
	3	Open Input/Output.
	9	Locked.
[39:08]		Language of program opening the file; same values as in the MCP language table.

Field	Value	Meaning
[31:04]		Type of access to use on file.
	0	Sequential.
	1	Random.
	2	Dynamic.
[27:01]		Deleted record flag.
	0	Deleted records are not visible.
	1	Deleted records are visible.
[26:01]		Presence of relative keys.
	0	No relative keys.
	1	Relative keys.
[25:01]		Record Units.
	0	Words.
	1	Characters.
[15:08]		Relative index into FILEINFO of first key of first key attribute (in words).
[19:04]		Record length flag.
	0	Fixed.
	1	Variable.
[07:08]		Number of keys.
[first key]		For the number of keys, list of the keys in ISAMKEYS attribute format.

OPENTYPE — Specifies how the file is to be opened. OPENTYPE has the same value as the FILEUSE attribute.

Value	Meaning
1	Input.
2	Output.
3	Input/Output.

The open result is an index into a table containing information about the user's state and position within the file.

<u>ISAMCLOSE</u>

ISAMCLOSE flushes buffers, unlocks any remaining locks (present due to abnormal termination), and closes the file.

The ISAMCLOSE procedure requires the following parameters and returns a close result:

ISAMCLOSE (CLOSETYPE, MYINFO)

CLOSETYPE	Specifies how the file is to be closed.
MYINFO	An index into a table describing the current state of the user's file. MYINFO is the value returned by ISAMOPEN.

The close result is an indication that all cleanup has been done in ISAMCLOSE.

ISAMSTART

ISAMSTART positions the user's current record pointer to the logical record currently in the file whose key satisfies the comparison. If the comparison is not satisfied by any record in the file, a "record not found" result is returned (bits 0 and 9 SET). A successful start operation establishes the RECORDKEY parameter as the key of reference; (the key of reference is the key that is used on any subsequent sequential operations). No data is transferred during a start operation.

The ISAMSTART procedure requires the following parameters and returns a start result:

ISAMSTART (RECORDKEY, KEYLENGTH, RECORD, CHOOZE, MYINFO)

RECORDKEY	Specifies the key to be used for the start operation. If the KEYFLAGF (bit 46) is SET, a keyed start is indicated, and RECORDKEY specifies the key in the same format as the ISAMKEYS attribute. If the KEYFLAGF is not SET, a relative start is indicated and the value contained in RECORDKEY is used as the relative key.
KEYLENGTH	The length of the key to be used in the start operation. KEYLENGTH must be less than or equal to the key-length field of the RECORDKEY. If KEYLENGTH is less than the key-length field of the RECORDKEY, a partial key start is indicated.
RECORD	The user's record area.

CHOOZE	Specifies the type of start to be done, as indicated in the following table:	
	Value	**Meaning**
	0	Start equal. (Finds the key with the same value as the key in the user's record.)
	15	Start greater than or equal. (Finds the first key with a value greater than or equal to the key in the key in the user's record.)
	20	Start greater than. (Finds the first key with a value greater than the key in the user's record.)
MYINFO	An index into a table describing the current state of the user's file. MYINFO is the value returned by ISAMOPEN.	

The start result values are listed in the following table:

Field	Value	Meaning
[0:01]	1	Error occurred.
	0	No error occurred.
[9:01]	1	No key met conditions.

ISAMSEQUENTIALWRITE

ISAMSEQUENTIALWRITE updates the file with the user's record and updates all key tables. Recovery information is saved before the key tables are updated. A write physically updates the keyed file. ISAMSEQUENTIALWRITE does not affect the current record pointer.

The ISAMSEQUENTIALWRITE procedure requires the following parameters and returns a sequential write result:

```
ISAMSEQUENTIALWRITE(RECORDKEY, RECORDLENGTH, RECORD, CHOOZE, MYINFO)
```

RECORDKEY	Not used for sequential writes.
RECORDLENGTH	The length of the record to be written.
RECORD	The user's record area.
CHOOZE	Not used.
MYINFO	An index into a table describing the current state of the user's file. MYINFO is the value returned by ISAMOPEN.

The sequential write result values are listed in the following table:

Field	Value	Meaning
[0:01]	0	No errors.
	1	Error occurred.
[5:01]	1	Duplicate key found and duplicates not allowed.
[6:01]	1	Primary keys are not in sequential order.

ISAMSEQUENTIALREAD

ISAMSEQUENTIALREAD reads the record specified by the current record pointer. The current record pointer is then updated to point to the next record in the file.

The ISAMSEQUENTIALREAD procedure requires the following parameters and returns a sequential read result:

```
ISAMSEQUENTIALREAD(RECORDKEY, RECORDLENGTH, RECORD, CHOOZE, MYINFO)
```

RECORDKEY	Not used for sequential reads.
RECORDLENGTH	The length of the record to be read.
RECORD	The user's record area.
CHOOZE	Not used.
MYINFO	An index into a table describing the current state of the user's file. MYINFO is the value returned by ISAMOPEN.

The sequential read result values are listed in the following table:

Field	Value	Meaning
[0:01]	0	No errors.
	1	Error occurred.
[9:01]	1	End of file.

ISAMRANDOMWRITE

ISAMRANDOMWRITE physically updates the KEYEDIO file with the user's record and updates all key tables. Recovery information is saved before the key tables are updated. ISAMRANDOMWRITE does not affect the current record pointer.

The ISAMRANDOMWRITE procedure requires the following parameters and returns a random write result:

```
ISAMRANDOMWRITE (RECORDKEY, RECORDLENGTH, RECORD, CHOOZE, MYINFO)
```

RECORDKEY	Specifies the key to be used for the write. If the KEYFLAGF (bit 46) is SET, a keyed write is indicated, and RECORDKEY has the same format as the ISAMKEYS attribute. If the KEYFLAGF is RESET, a relative write is indicated, and the value contained in RECORDKEY is used as the relative key.
RECORDLENGTH	The length of the record to be written.
RECORD	The user's record area.
CHOOZE	Not used.
MYINFO	An index into a table describing the current state of the user's file. MYINFO is the value returned by ISAMOPEN.

The random write result values are listed in the following table:

Field	Value	Meaning
[0:1]	0	No errors.
	1	Error occurred.
[5:1]	1	Duplicate key found and duplicates not allowed.

ISAMRANDOMREAD

ISAMRANDOMREAD reads the record specified by RECORDKEY. The key specified by RECORDKEY becomes the key of reference, and the current record pointer is updated to point to the next record.

The ISAMRANDOMREAD procedure requires the following parameters and returns a random read result:

```
ISAMRANDOMREAD(RECORDKEY, RECORDLENGTH, RECORD, CHOOZE, MYINFO)
```

RECORDKEY	Specifies the key to be used for the read. If the KEYFLAGF (bit 46) is SET, a keyed read is indicated, and RECORDKEY has the same format as the ISAMKEYS attribute. If the KEYFLAGF is RESET, a relative read is indicated, and the value contained in RECORDKEY is used as the relative key.
RECORDLENGTH	The length of the record to be read.
RECORD	The user's record area.
CHOOZE	Not used.
MYINFO	An index into a table describing the current state of the user's file. MYINFO is the value returned by ISAMOPEN.

The random read result values are listed in the following table:

Field	Value	Meaning
[0:01]	0	No errors.
	1	Error occurred.
[9:01]	1	Record not found.

ISAMREWRITE

ISAMREWRITE rewrites the record specified by RECORDKEY. If a serial rewrite is done, the next record specified by the current record pointer is rewritten. If a random rewrite is done, RECORDKEY specifies the record to be rewritten. ISAMREWRITE does not affect the current record pointer.

The ISAMREWRITE procedure requires the following parameters and returns a rewrite result:

```
ISAMREWRITE(RECORDKEY, RECORDLENGTH, RECORD, CHOOZE, MYINFO)
```

RECORDKEY	Specifies the key to use for the rewrite. If the KEYFLAGF (bit 46) is SET, a keyed rewrite is indicated. RECORDKEY then has the same format as the ISAMKEYS attribute. If KEYFLAGF is RESET, a relative write is indicated, and the value contained in RECORDKEY is used as the relative key.
RECORDLENGTH	The length of the record to be rewritten.
RECORD	The user's record area.
CHOOZE	Specifies whether a serial rewrite or a sequential rewrite is to be done.

Value	Meaning
16	Rewrite serial. File must be opened with ACCESSMODE sequential or dynamic, and the previous operation on the file must have been a successful read.
18	Rewrite random. Replaces the record specified by RECORDKEY with the record contained in RECORD.

MYINFO	An index into a table describing the current state of the user's file. MYINFO is the value returned by ISAMOPEN.

The rewrite result values are listed in the following table:

Field	Value	Meaning
[0:01]	0	No errors.
	1	Error occurred.
[6:01]	1	Primary keys are not equal.
[9:01]	1	Record not found.

ISAMDELETE

ISAMDELETE deletes the specified record. The previous I/O operation must have been a successful read if a serial delete is to be done. A serial delete deletes the record previously read. A random delete deletes the record specified by RECORDKEY. ISAMDELETE does not affect the current record pointer.

The ISAMDELETE procedure requires the following parameters and returns a delete result:

```
ISAMDELETE(RECORDKEY, RECORDLENGTH, RECORD, CHOOZE, MYINFO)
```

RECORDKEY	Specifies the key to be used for the delete. If the KEYFLAGF is SET, a keyed delete is indicated, and RECORDKEY has the same format as the ISAMKEYS attribute. If the KEYFLAGF is RESET, a relative delete is indicated, and the value contained in RECORDKEY is used as the relative key.
RECORDLENGTH	The length of the record to be deleted.
RECORD	The user's record area.
CHOOZE	Specifies whether a serial or random delete is to be done.

Value	Meaning
17	Delete serial; deletes the record previously read. File must be opened with ACCESSMODE sequential or dynamic, and the previous operation must have been a successful sequential read.
19	Delete random; deletes the record specified by RECORDKEY.

MYINFO	An index into a table describing the current state of the user's file. MYINFO is the value returned by ISAMOPEN.

The delete result values are listed in the following table:

Field	Value	Meaning
[0:01]	0	No errors.
	1	Error occurred.
[6:01]	1	Primary keys are not equal.
[9:01]	1	Record not found.

SETUPPERLIMIT

SETUPPERLIMIT defines the upper bounds of the file. This logical end-of-file (EOF) is set only for the key defined by RECORDKEY. The user's RECORD contains the value of the upper bound. If UPPERLIMIT is set and an attempt is made to access beyond this limit, an end-of-file (eof) condition is returned.

The SETUPPERLIMIT procedure requires the following parameters and returns a set-upper-limit result:

```
SETUPPERLIMIT (RECORDKEY, RECORDLENGTH, RECORD, CHOOZE, MYINFO)
```

RECORDKEY	Specifies the key whose logical end-of-file is set. If the KEYFLAGF (bit 46) is SET, a set-upper-limit for a KEYEDIO key is indicated, and RECORDKEY has the same format as the ISAMKEYS attribute. If the KEYFLAGF is RESET, the value contained in RECORDKEY is used as the upper limit of the relative keys.
RECORDLENGTH	Used as the length of the key. RECORDLENGTH is used in place of the KEYLENGTHF field of RECORDKEY.
RECORD	The user's record area.
CHOOZE	Must have value of 21 (set upper limit) in [6:6].
MYINFO	An index into a table describing the current state of the user's file. MYINFO is the value returned by ISAMOPEN.

The set-upper-limit result values are listed in the following table:

Field	Value	Meaning
[9:01]	1	RECORDKEY is invalid.

ISAMPWRITEN

ISAMPWRITEN performs a random write, delete, or rewrite operation. ISAMPWRITEN calls the appropriate I/O routine to handle the operation specified by CHOOZE.

The ISAMPWRITEN procedure requires the following parameters and returns a PWRITEN result:

ISAMPWRITEN (RECORDKEY, RECORDLENGTH, RECORD, CHOOZE, MYINFO)

RECORDKEY	Specifies the key to be used for the update. If KEYFLAGF (bit 46) is SET, a keyed update is indicated, and RECORDKEY has the same format as the ISAMKEYS attribute. If KEYFLAGF is RESET, a relative update is indicated, and the value specified by RECORDKEY is used as the relative key.
RECORDLENGTH	The length of the record to be read.
RECORD	The user's record area.
CHOOZE	Specifies the type of operation to be done.

Value	Meaning
16	Rewrite serial; performs a rewrite operation, updating the record specified by the current record pointer.
17	Delete serial; deletes the record specified by the current record pointer.
18	Rewrite random; updates the record specified by RECORDKEY.

Value	Meaning
19	Delete random; deletes the record specified by RECORDKEY.
21	Set upper limit; sets the logical end-of-file for the key specified by RECORDKEY to the key value in RECORD.

MYINFO	An index into a table describing the current state of the user's file. MYINFO is the value returned by ISAMOPEN.

The PWRITEN result values depend on the type of operation defined in the CHOOZE word.

7.6 THE KEYEDIO FILE

The following subsection describes the structure of the KEYEDIO file.

SEGMENT 0 (ZERO) OF THE FILE

When a KEYEDIO file is created, the following information about the file is saved in segment 0 of the file. Words in segment 0 are specified as offsets from the standard block information area at the start of every block. Since the block information area is currently 10 words long, word 0 of segment 0 is actually located at word 10 of the block.

Word 0 Describes the physical characteristics of the file. The values stored in this word are calculated at file creation time based on the user's file declarations and space requirements needed for maintaining the file.

Field	Meaning
[47:08]	The offset into the recovery area where the contents of the new record are stored for recovery of a rewrite operation.
[39:08]	The default number of buffers to allocate when this file is opened. This default is established by the setting of the BUFFERS attribute when the file was created.
[31:16]	The size of the records contained in the file in words. This value may be different from the user's declared MAXRECSIZE because of relative keys. If the user program has specified relative keys, the record size is adjusted to allow relative keys.
[15:16]	The block size of the file in words. This value is different from the user's declared block size.

Word 1 Contains information about the keys and the program that created the file. This word is copied directly from FILEINFO [0] in ISAMOPEN at file creation time.

Field	Value	Meaning
[47:04]		File format level (currently 1).
[43:04]		Status of file.
	0	Closed.
	1	Open Input.
	2	Open Output.
	3	Open Input/Output.
	9	Locked.
[39:08]		Language of program opening the file; same values as in the MCP language table.
[31:04]		Type of access to use on file.
	0	Sequential.
	1	Random.
	2	Dynamic.
[27:01]		Deleted record flag.
	0	Deleted records are not visible.
	1	Deleted records are visible.
[26:01]		Presence of relative keys.
	0	No relative keys.
	1	Relative keys.
[25:01]		Record Units.
	0	Words.
	1	Characters.
[15:08]		Relative index into FILEINFO of first key of first key attribute (in words).
[19:04]		Record length flag.
	0	Fixed.
	1	Variable.

[07:08]	Number of keys.
[first key]	For the number of keys, list of the keys in ISAMKEYS attribute format.

Word 2 Specifies the timestamp of the last update. The timestamp is used in recovery. The current value of TIME (6) is stored in this word whenever the file is about to be updated. Every block that is written because of the update contains this timestamp. (Refer to the "Recovery" section.)

Word 3 Specifies the segment that describes the keys (the key information table).

Field	Meaning
[39:20]	Length of the key information.
[19:20]	Relative segment number of key information block.

Word 4 Not currently used.

Word 5 Specifies the first block in the file that has never been used. All blocks beyond the specified one are available for use.

Field	Meaning
[43:24]	Relative segment number of the first available block.

Word 6 Specifies the next available record slot. When adding a record, the new record is stored at the location specified by this word.

Field	Meaning
[43:24]	The relative segment number of the current block.
[19:20]	The offset into the block of the next record location.

Word 7	Specifies the last user record to be updated. This word is only valid if non-zero. (Refer to the "Recovery" section.)

Field	Meaning
[47:04]	The type of update in process.
[43:24]	The relative segment number of the last updated block.
[19:20]	The offset into the block of the last updated record.

Word 8	The next relative key to be allocated. If relative keys have been specified, this word contains the next relative key value to be allocated.
Word 9	User specified MINRECSIZE when the file was created.
Word 10	User specified MAXRECSIZE when the file was created.
Word 11	User specified BLOCKSIZE when the file was created.
Word 12	User specified FRAMESIZE when the file was created.
Word 13	User specified BLOCKSTRUCTURE when the file was created.
Word 14	User specified UNITS when the file was created.
Word 15	User specified EXTMODE when the file was created.

See also

Recovery. 268

BLOCK INFORMATION LAYOUT

A keyed file is made up of data and tables. Data and tables have the same size, and each block contains the following control information:

Word 0 Contains information about the block, table organization, and key size.

Field	Value	Meaning
[47:04]		Type of block.
	1	Coarse table.
	2	Fine table.
	3	Data block.
[43:01]		Table organization (only valid for coarse and fine tables).
	0	Descending order by key.
	1	Ascending order by key.
[15:16]		Size of each key entry (only valid for coarse and fine tables)

Word 1 Contains information about the keys (only valid for coarse and fine tables).

Field	Meaning
[47:16]	First key entry (word offset).
[31:16]	Number of keys currently in table.
[15:16]	Maximum number of key entries that can fit in the table.

Word 2 Address (relative segment number) of this block.

Word 3 Address (relative segment number) of the next sequential fine table. This word links fine tables together so that the file can be accessed sequentially. (Only valid if this is a fine table block.)

Word 4 Address (relative segment number) of the previous sequential fine table. This word maintains the sequential ordering of fine tables. (Only valid if this is a fine table block.)

Word 5 Timestamp of the block. This word is used in recovery.

COARSE TABLE LAYOUT

Coarse tables are made up of keys and table pointers. The key value is the value of the last key contained in the block specified by the table pointer. The table pointers are relative segment numbers into the file of the next table down in the structure. Table pointers are aligned on word boundaries.

Figure 7-1 describes a coarse table layout.

```
              |<-SIZE OF EACH   |
              |   KEY  ENTRY--->|
|=====================================================================|
|             |     |           |     |            |     |           |
| BLOCK INFO  | KEY | TABLE PTR | KEY | TABLE   PTR| KEY | TABLE PTR |
|             |     |           |     |            |     |           |
|=====================================================================|
```

Figure 7-1. Coarse Table Layout

FINE TABLE LAYOUT

Fine tables are made up of keys and pointers. The key value is the same value that the record specified by the data pointer contains. The pointers are made up relative segment numbers into the file of the data block and a word offset of where the data starts within the block. The data pointers have the following format:

Field	Meaning
[43:24]	The relative segment number of the block that contains the record.
[19:20]	The word offset into the block of the beginning of the record.

Figure 7-2 describes a fine table layout.

```
                 |<-SIZE OF EACH   |
                 |   KEY  ENTRY--->|
|=====================================================================|
|            |      |            |     |            |     |           |
|            |      | DATA   PTR |     | DATA   PTR |     | DATA   PTR |
|            |      |            |     |            |     |           |
| BLOCK INFO | KEY  | (segment)  | KEY | (segment)  | KEY | (segment) |
|            |      | (offset)   |     | (offset)   |     | (offset)  |
|            |      |            |     |            |     |           |
|=====================================================================|
```

Figure 7-2. Fine Table Layout

KEY INFO TABLE LAYOUT

The key information table contains 3 word entries that describe each key in the file and the first tables to be used in accessing the file with that key.

The first word of each key information table entry describes the key. It contains the same information in the same format as used for the ISAMKEYS file attribute discussed under "Indexed (KEYEDIO) File Attributes" .

The second word of each entry contains the relative segment address of the root table of the index tables for this key. If all the entries for this key will fit into a single table, the root table will be a fine table. Otherwise it will be a coarse table. The root table is the highest block in the hierarchical structure of index tables; it is the first table to be searched when accessing a file randomly.

The third word of each key information table entry contains the relative segment address of the first fine table for this key. It is used to find the first record when accessing the file sequentially.

See also

Indexed (Keyedio) File Attributes 224

LOGICAL LAYOUT OF FILE

The logical layout of a KEYEDIO file is illustrated in Figure 7-3.

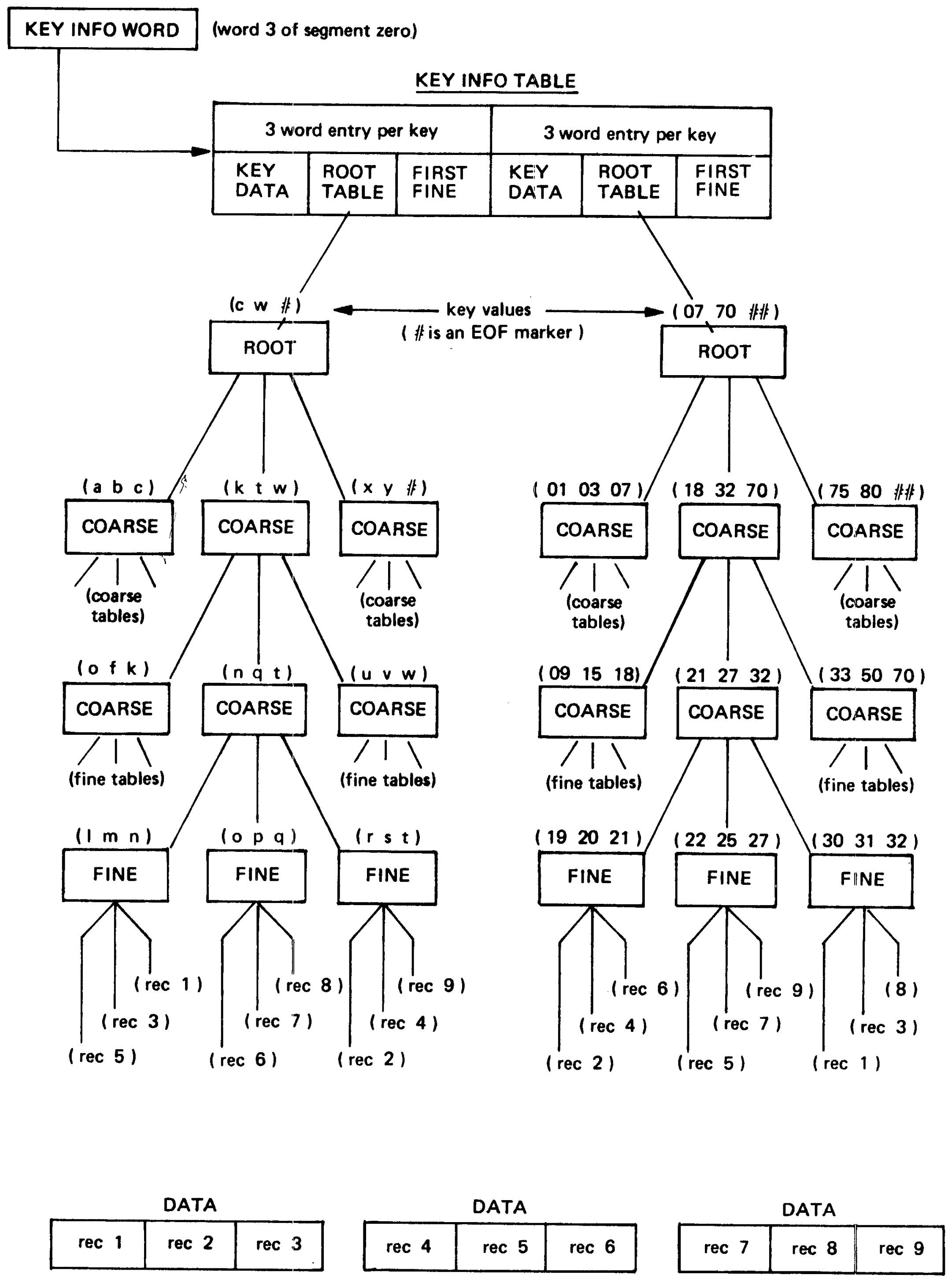


Figure 7-3. Logical Layout of File

The end-of-file (EOF) marker is actually a key entry of all bits SET and a data pointer of zero.

```
                          KEY INFORMATION            ROOT PTR   1st FINE
                  |---------------------------------------------------------|
                  | key flag          = 1         |         |             |
                  | alternate key     = FALSE     |         |             |
                  | duplicate keys    = TRUE      |         |             |
KEY INFO TABLE    | key organization  = ascending |    5    |     21      |
------------------| key sign          = no sign   |    .    |             |
(SEGMENT #1)      | key type          = EBCDIC    |    |    |             |
                  | keylength         = 7         |    |    |             |
                  | keyoffset         = 3         |    |    |             |
                  |-------------------------------|---|----|------------|
                                                       |
                                                       |
       ------------------------------------------------|
       |
       |---->|----------------------------------------------------------|
             |       |         |    |          |    |          |    |
 ROOT TABLE  | BLOCK |         |    |          |    |          |    |
------------ | INFO  | JOHNSON | 9  | WILLIAM  | 91 | #######  | 0  |
 SEGMENT #5  |-------|---------|-|--|----------|----|----------|----|
       ---------------------------|       note:
       |                                  '#######'is the EOF marker
       |---->|----------------------------------------------------------|
             |       |         |    |          |    |          |    |
COARSE TABLE | BLOCK |         |    |          |    |          |    |
------------ | INFO  |  BAKER  | 19 |   HILL   | 23 | JOHNSON  | 35 |
 SEGMENT #9  |-------|---------|-|--|----------|----|----------|----|
       ---------------------------|
       |
       |---->|----------------------------------------------------------|
             |       |         |    |          |    |          |    |
COARSE TABLE | BLOCK |         |    |          |    |          |    |
------------ | INFO  |  ALLEN  | 21 | ANDREWS  | 57 |  BAKER   | 47 |
SEGMENT #19  |-------|---------|----|----------|-|--|----------|----|
       ------------------------------------------|
       |
       |--->|-----------------------------------------------------------|
            |       |        |              |         |              |
 FINE TABLE | BLOCK |        | SEGMENT 111  |         | SEGMENT 321  |
----------- | INFO  | ANDERS | OFFSET    23 | ANDREWS | OFFSET    36 |
SEGMENT #57 |-------|--------|--------|----|---------|--------------|
      -------------------------------|
      |            WORD                    WORD
      |             10                      23
      |---->|-----------------------------------------------------------|
            |       |                       |                        |
 DATA BLOCK | BLOCK |                       |                        |
----------- | INFO  |  001ABCDEFG  12345    |  002ANDERS  34567      |
(SEGMENT #111)|-------|-----------------------|------------------------|
```

Figure 7-4. Example of Locating Data

INSERTING KEYS

Information contained in the block information is used for determining how to add the new key. The first entry in the table, the number of entries currently in the table, and the maximum number of entries that can fit in the table are saved in the block information. If the number of entries is less than the maximum number of entries, keys to the left of the new key are moved to the left, and the new key is inserted. To add a new key entry to a table that is already full, the table is split. When the split is made, all entries to the left of the new key and the new key go into one table, and all entries to the right go into another table.

NOTE

The tables are always right-justified.

An Example of Inserting a Key:

```
                |<-------KEY-------->|<--DATA PTR->|
                |==================================|
        FINE    |                    |             |
        TABLE   |       BBBBBB       |     137     |
        KEY     |                    |      23     |
        ENTRY   |                    |             |
                |==================================|
added to
```

```
                |<UNUSED AREA  FOR|
        |<BLOCK |   ADDITIONAL    |<---KEY-->|<DATA|<---KEY-->|<DATA|
        |  INFO>|  KEY   ENTRIES> |          | PTR>|          | PTR>|
        |===========================================================|
        |       |          |      |          |     |          |     |
FINE    | INFO  | //////// | //// |  AAAAAA  | 157 |  CCCCCC  | 147 |
TABLE   |       | //////// | //// |          |  10 |          |  36 |
        |===========================================================|
becomes
```

```
        |<BLOCK |<---KEY-->|<DATA |<---KEY-->|<DATA|<---KEY-->|<DATA|
        |  INFO>|          | PTR> |          | PTR>|          | PTR>|
        |===========================================================|
        |       |          |      |          |     |          |     |
FINE    | INFO  |  AAAAAA  | 157  |  BBBBBB  | 137 |  CCCCCC  | 147 |
TABLE   |       |          |  10  |          |  23 |          |  36 |
        |===========================================================|
```

An Example of Inserting a Key into a Full Table:

```
                |<---KEY-->|<DATA|
                |          | PTR>|
                |================|
        FINE    |          |     |
        TABLE   |  AAABBB  | 173 |
        ENTRY   |          |  23 |
                |================|
added to

           |<BLOCK |<---KEY-->|<DATA|<---KEY-->|<DATA|<---KEY-->|<DATA|
           | INFO> |          | PTR>|          | PTR>|          | PTR>|
           |=========================================================|
           |       |          |     |          |     |          |     |
   FINE    | INFO  | AAAAAA   | 157 |  BBBBBB  | 137 |  CCCCCC  | 147 |
   TABLE   |       |          |  10 |          |  23 |          |  36 |
           |=========================================================|
            (SEGMENT 10)

becomes

                 |<-UNUSED AREA    |
          |<BLOCK |FOR  ADDITIONAL  |<---KEY-->|<TABLE|<---KEY-->|<TABLE|
          | INFO> | KEY  ENTRIES -> |          | PTR> |          | PTR> |
          |==============================================================|
          |       |         |       |          |      |          |      |
   COARSE | INFO  | /////// | ///   |  AAABBB  |  15  |  CCCCCC  |  51  |
   TABLE  |       | /////// | ///   |          |      |          |      |
          |==============================================================|
           (SEGMENT 14)
```

KEYEDIO

```
                 |<-UNUSED AREA   |
        |<BLOCK |FOR  ADDITIONAL  |<---KEY-->|<DATA  |<---KEY-->|<DATA |
        |  INFO>| KEY  ENTRIES -> |          | PTR>  |          | PTR>  |
        |==============================================================|
        |       |          |      |          |       |          |       |
FINE    | INFO  | ///////  | ///  |  AAAAAA  | 157   |  AAABBB  | 173   |
TABLE   |       | ///////  | ///  |          |  10   |          |  23   |
        |==============================================================|
          (SEGMENT 15)


                 |<-UNUSED  AREA  |
        |<BLOCK |FOR  ADDITIONAL  |<---KEY-->|<DATA  |<---KEY-->|<DATA |
        |  INFO>| KEY  ENTRIES -> |          | PTR>  |          | PTR>  |
        |==============================================================|
        |       |          |      |          |       |          |       |
FINE    | INFO  | //////// | ///  |  BBBBBB  | 137   |  CCCCCC  | 147   |
TABLE   |       | //////// | ///  |          |  23   |          |  36   |
        |==============================================================|
          (SEGMENT 51)
```

7.7 RECOVERY

To ensure that the file is always in a consistent state, recovery information is saved before any updating is done to the file. The recovery information consists of a timestamp and a pointer to the record that is being updated. Recovery information is not saved when the file is being created.

The following sequence of events takes place when a file is updated:

1. The recovery information is saved in segment 0 of the file.

2. The writes to the file are done in a careful order so that information is not lost if the writes are not completed. The data record affected by the update is the last record changed.

If the update is terminated before step 2 is finished, the update is completed the next time the file is opened. The recovery process begins when the record referenced by the recovery information in segment zero is read. The file is then updated with that record. During the update process, if a block is read that has a timestamp equal to that of the recovery timestamp, that block is treated as already reflecting the update. After the update is done, the recovery information is zeroed out, and segment zero is rewritten.

RECOVERY MESSAGES AND WARNINGS

The following warning message is displayed when a KEYEDIO file has Mark 3.3 PR1 or earlier recovery information:

```
FILE TOO OLD FOR HALTLOAD RECOVERY.
FILE MUST BE RE-CREATED WITH 3.4.1 OR LATER KEYEDIO.
```

The recovery can still be attempted, if necessary. If KEYEDIO is unable to begin recovery with the information in the file, the following message is displayed:

```
INSUFFICIENT RECOVERY STATE.
```

If there is sufficient information to begin recovery but the recovery cannot be completed, the following message is displayed:

```
UNABLE TO RECOVER FILE.
```

For both of these conditions, the user also receives the following message:

```
WARNING: FILE MAY BE CORRUPTED.
THIS FILE SHOULD BE RELOADED USING 3.4.1 OR
LATER KEYEDIO.
'HI' TO CONTINUE, 'DS' TO ABORT.
```

Enter HI to resume processing and terminate the recovery attempt. Enter DS to discontinue the application.