HONEYWELL

MULTICS SYSTEM DIAGNOSTIC AIDS

SOFTWARE

MULTICS SYSTEM DIAGNOSTIC AIDS

SUBJECT

Description of Standard Multics Commands and Facilities to Be Used in Diagnosing System Malfunctions

SPECIAL INSTRUCTIONS

This revision supersedes Revision 2 of the manual, whose title was *Multics Hardware Diagnostic Aids*, dated July 1982. Sections 2 and 3 have been merged and their commands sorted. The title of the new section has been changed to "Commands." Section 4 was renamed to "Section 3."

Marginal change indicators (change bars and asterisks) indicate technical changes. New commands do not contain change bars.

SOFTWARE SUPPORTED

Multics Software Release 10.2

ORDER NUMBER

AR97-03

December 1983

Honeywell

PREFACE

This manual contains information describing various system diagnostic aids that are provided as part of Multics.

You should be familiar with some of the concepts and terminology of the Multics system and the hardware on which it runs. Throughout this manual, reference is frequently made to a number of the following manuals:

Manual Name	Order No.	Text Reference
Multics Online Test and Diagnostics Reference Manual	*AU77*	*Online T&D*
Multics Programmer's Reference Manual	*AG91*	*Reference Manual*
Multics Commands and Active Functions	*AG92*	*Commands*
Multics Subroutines and I/O Modules	*AG93*	*Subroutines*
Multics Operator's Handbook	*AM81*	*MOH*
Multics Processor Reference Manual	*AL39*	*Processor Manual*
Multics Bulk Input/Output	*CC34*	*Bulk I/O*

SIGNIFICANT CHANGES IN AR97-03

There are two new commands: analyze_multics and display_cpu_error. The following commands have had changes: dump_mpc, et (now named "eis_tester, et"), io_error_summary, mos_edac_summary, and print_syserr_log. The check_cpu_speed command has been completely revised.

File No.: 1L13, 1U13

AR97-03

CONTENTS

SECTION 1

CONTENTS OF THE SYSERR LOG

DESCRIPTION OF LOG

The syserr log contains all messages that are printed on the operator console by Multics and the storage system salvager. There are also messages of the same class that are not printed on the operator console. Tools that exist to peruse and print the contents of this log are described in Section 2.

The syserr log is located in a special area of disk storage that is specified to the system via a PART LOG configuration card. Refer to the MOH for more information concerning the PART LOG configuration card. *

The general format of the syserr log consists of a header followed by a list of threaded messages. The format of individual messages in the log is described below.

MESSAGE FORMAT

A syserr message consists of the following components:

sequence number
: A number that is increased by one for each syserr message. This generally serves to number the messages, but sequence numbers may be missing, or they may restart at 0 at any point in the log.

time
: The date and time when the message was logged.

code
: This is a combined sorting class and action code. The low-order decimal digit is the syserr action code. The high-order digits are the sorting class. Both are explained below.

text
: This is the text of the message.

binary data
: An optional field that is used by some error messages to log more detailed information about the cause of the error than could be explained in the text.

The syserr action codes (the low-order digit of the code above) represent the action that was taken at the time the message occurred. These are summarized below.

0 print message on console, and log it.

1 ring alarm, print message on console, and crash system. (Since the system crashes immediately, these messages do not generally find their way into the log.)

2 ring alarm, print message on console, log it, and terminate the process issuing the message.

3 ring alarm, print message on console, and log it.

4 log message only, but print it if there is no room in the wired syserr buffer.

5 log message only, but discard it if there is no room in the wired syserr buffer.

6-9 not used.

The syserr sort codes are used to indicate special classes of messages. Almost all messages currently fall into class 0, including all hardware-related messages.

EXAMPLE OF LOG CONTENTS

The following is a sample of the syserr log contents as it would be printed by print_syserr_log, or daily_syserr_process. (See Section 2 for an explanation of these commands.)

```
Sat  04/24/76 (est)
1310.8    34548   0 added memory e
1313.2    34549   0 added cpu b
1313.7    34550   0 RCP: Attached tape_06 for Dumper.SysDaemon.z
1313.7    34551   3 RCP: Mount Reel 50202 without ring on tape_06
                      for Dumper.SysDaemon.z
1313.7    34552   4 RCP: Assigned tape_06 to Dumper.SysDaemon.z
1314.9    34553   0 RCP: Detached tape_06 from Dumper.SysDaemon.z
1421.0    34554   0 disk_control: dev attention for dskb 4
                    (iom 1 chan 35) cmd 25 stat 422000000100
1421.0    34555   0 disk_control: dskb 4 sect=317740 cyl=214 hd=1
                                  addr=10120
1421.0    34556   0 disk_control: dskb 4 detailed status =
                      40 00 00 00 81 00 00 00 00
1421.0    34557   3 disk_control: dskb 4 requires intervention
1430.4    34558   4 RCP: Assigned tape_02 to Fred.AvgUser.m
1430.5    34559   0 RCP: Attached tape_02 for Fred.AvgUser.m
1430.5    34560   0 RCP: Note (tape_02) - 50202,den=1600
1430.5    34561   3 RCP: Mount Reel 50202 with ring on tape_02 for
                      Fred.AvgUser.m
1431.3    34562   0 RCP: Detached tape_02 from Fred.AvgUser.m
1432.2    34563   0 hardware fault: parity fault on CPU C by
                                    Herbie.SysMaint
1432.6    34564   4 mos_memory.check: EDAC error on mem c
  ^         ^     ^ ^
  |         |     | |
  |         |     | +-->  Text of message.
  |         |     |
  |         |     +-->  syserr code.
  |         +->   sequence number.
  |
  +--> Time message logged.
```

DESCRIPTION OF HARDWARE ERROR MESSAGES

For convenience, the conventions listed below are used in the error messages shown in the following paragraphs.

D	represents a decimal number
N	represents an octal number
W	represents a full word, in octal

I/O Errors

I/O errors are logged in several different ways, depending on the device. They are classified here into several groups. Each is explained in separate paragraphs.

1. User devices. This group includes all unit record devices (printers, readers, punches), all tape drives, and I/O disks (disks not used by the storage system).

2. Storage system disks.

3. The operator console.

4. The input/output multiplexer (IOM).

*

I/O Errors for User Devices

Errors for these devices are indicated by a message (as printed by print_syserr_log with the -expand control argument) of the following format:

ioi_interrupt: I/O error. iom=N1 chan=N2 device=N3 status=W

where:

1. N1 is the IOM number.

2. N2 is the channel number.

3. N3 is the device number

4. W is the first IOM status word.

Additional messages that occur each time a user device is attached also appear in the log. For example:

RCP: Errors (dev) = n.

where:

1. dev is the name of the device just detached.

2. n is the number of errors that occurred during the attachment.

It should be noted, however, that this error count is generated by a user ring program that may or may not count errors, and if it does count them, may or may not count them correctly.

See Section 2 for a description of the io_error_summary command, which scans the log and summarizes I/O errors.

Disk Errors

Errors encountered by operation of the storage system on disks result in messages in the following formats:

disk_control: Queuing error
an internal coding error is given.

disk_control: Reconnected DSKX_NN (iom X chn YY)
where "DSKX_NN" is the disk subsystem name, such as "dska"; "NN" is the drive number, in decimal; "X" is the IOM number, in octal; and "YY" is the channel number, in octal. A disk interrupt was apparently lost. Status for the disk did not arrive within the expected time. The system restarts the disk operation.

disk_control: Placing DSKX_NN in operation
a special interrupt has been received for a disk drive marked as broken. The system attempts to use the device.

disk_control: Unexpected IOM status SSSS for DSKX_NN (iom X chn YY)
status has been received from a channel that was not marked active. This is due to a disk subsystem or IOM problem, or to a logic error in the supervisor. The system ignores the status and attempts to continue operation. See the *Hardware and Software Formats, PLM* manual (Order No. AN87) for an interpretation of the SSSS status.

disk_control: DSKX_NN now operational
a disk drive that required intervention has successsfully completed an I/O operation. The system again uses its contents.

disk_control: DSKX_NN requires intervention
a disk error has occurred that could have been caused by the pack or drive being broken or requiring operator attention. The system has retried the operation several times without success. The system will try the device periodically to check if it has been repaired.

disk_control: MAJOR_STAT SUBSTAT for DSKX_NN (iom X chn YY)
where "STAT" is the name of major status and substatus, in character form, such as "dev attention." A disk error has occurred on drive DSKX_NN. The major status and substatus are interpreted as character strings. The disk address is given both as a Multics record address in octal (RRRR) and as an absolute sector number in octal (SSSS). The main store address being used was AAAA. The hexadecimal value of the detailed status is given in cases where this data is useful. See the *Hardware and Software Formats, PLM* manual (Order No. AN87) for an interpretation of this information.

disk_control: Removing channel YY on IOM X
errors occurred are indicative of a defective disk channel or MPC. The channel receiving the errors is placed offline.

Operator Console Errors

Errors encountered in operation of the operator console result in a message in the following format:

ocdcm_ (log_console_error): i/o error status = W1 flags = W2

where:

1. W1 is the IOM status word.

2. W2 is a word of software flags. The flags and switches with the specified meanings are

f0	active	ON => this entry is in use
f1	bootload_console	ON => this is the bootload console
f2	alternate	ON => console is alternate
f3	inop_device	ON => console is inoperative
f4	io_device	ON => console is an IOI usable device
f5	config_change	ON => console change has occurred
f6	prompt	ON => prompt for input
f7	pcw_io	ON => use PCWs instead of IDCWs
f8	io_in_progress	ON => I/O op is in progress
f9	got_special_int	ON => RE(TURN QUEST) key was hit
f10	oper_request	ON => operator has hit request key
f11	break	ON => suspend output

Input/Output Multiplexer Errors

General input/output multiplexer (IOM) errors result in a message in the following format:

iom_manager: system fault status W.

where W is an IOM system fault word.

If an I/O operation fails to complete within a reasonable time interval, a message in the following format is printed:

ioi_timer: Timeout on IOM N1, chan N2, dev N3

where:

1. N1 is the IOM number.

2. N2 is the channel on which the timeout occurred.

3. N3 is the device that was connected when the timeout occurred.

Main Memory or CPU Errors

Errors in the operation of the CPU that result in parity, op-not-complete, shutdown, or startup faults result in messages that have the format

hardware_fault: FFF fault on CPU TAG by User_id path

where:

1. FFF
   is a fault name.

2. TAG
   is the tag of the CPU on which the fault occurred. The value of TAG can be the letters a through h.

3. User_id
   is the name of the user whose process took the fault.

4. path
   is the pathname of the procedure in which the fault was taken.

Machine conditions and history registers are logged in the binary portion of the message. A sample printout of the parity fault, as would be provided by using the -expand control argument of print_syserr_log or daily_syserr_process, follows:

```
Fri  05/05/83 (edt)
1555.2  74663  0 hardware_fault: parity fault on CPU C by
Jones.Multics.a in >user_dir_dir_>Multics>Jones>mydoc.
      Pointer Registers:
      000244400043 013760000000 000116400043 000024000000
      000330400043 000000000000 000072400043 003711000000
      000253400043 034424000000 000253400043 034424000000
      000244400043 013460000000 000244400043 000000000000
      x0-7: 012222 013644 000000 000004 000007 000000 000016 000571
      a: 000000000000 q: 000000000000 e:000 t: 413500100 ralr: 0
      SCU data:
      400350170041 000003150023 400330000200 000000000000
      000006401200 000000000000 200000757100 000007000000
      Fault Register: 013000006400
      EIA Info:
      000400000000 000400000000 004314000000 000077777774
      000024000000 000077777774 004116000004 000000000077
      OU History Reg Data:
      757000757300 177777000005 237000237100 123777000001
      757000757300 177777000005 237000237100 123777000001
      757000757300 177777000005 237000237100 123777000001
      757000757300 177777000005 237000237100 123777000001
      757000757300 177777000005 237000237100 123777000001
      757000757300 177777000005 237000237100 123777000001
      757000757300 177777000005 237000237100 123777000001
      757000757300 177777000005 757000757000 037777000006
      CU History Reg DATA:
      200007011000 000000042011 600137011000 000004042201
      200127011000 000000002201 200007011000 000000042202
      600137757100 000006042200 200127757100 000000242041
      201037710000 000001042201 600137237100 000002042200
      200127237100 000000042011 200007011000 000000042011
      600137011000 000004042201 200127011000 000000002201
      200007011000 000000042202 600137757100 000006042200
      200127757100 000000242041 201005710000 000122010002
      DU History Reg DATA:
      777757037717 744243410015 777757037737 744242410015
      737757037737 744242410015 777757037737 744243410015
      737757037717 744243410015 777757037737 744243410015
      777757037737 744243410015 777757037737 744243410015
      737757037717 744243410015 777757037737 744243410015
      737757037737 744243410015 777757037737 744243410015
      777757037717 744243210015 777757037737 744243410015
      737757037737 744243410015 777757037737 744243410015
      APU History Reg Data:
      003302007252 004160004775 003506007150 004060044775
      003506007150 004060064775 003302007252 004160004775
      003506007150 004060014775 003506007150 004060024775
      003302007252 004160004775 003506007150 004060044775
      003506007150 004060014775 003506007150 004060024775
      003506007150 004060014775 003506007150 004060024775
      003506007150 004060014775 003506007150 004060024775
```

The other three faults are handled in a similar manner.

MOS Memory EDAC Errors

An EDAC error on a MOS memory is logged with a message (as printed by print_syserr_log with the -expand control argument) in the following format:

```
mos_memory_check: EDAC error on mem M MOS S chip, Error: board B,
  chip C
```

where:

1. M is the tag of the SC containing the bad memory where M is a letter from a to h. The error is in the store unit containing the first address in this SC.
2. S is the chip size, 1k or 4k.
3. B is the number of the board containing the bad chip.
4. C is the number of the bad chip on board B.

DATANET 6600 Front-End Network Processor (FNP) Errors

Certain errors in the operation of the DATANET 6600 Front-End Network Processor result in messages in the following formats:

```
3 system_control_:  A is hung up
3 system_control_:  A error W
```

where:

1. A is a device name of a terminal channel controlled by the message coordinator.
2. W is an error_table_ code.

Console Messages from FNP

Certain error conditions in the FNP cause messages to appear on the Multics operator console. Most of these conditions are sufficiently serious to cause the FNP to crash and generate a dump.

The general format of a crash message from the FNP is as follows:

```
emergency interrupt from 355 TAG: FAULT
  355 instruction counter = IC
```

where:

1. TAG identifies the FNP that crashed.
2. FAULT is the name of the fault that occurred in the FNP.

3. IC is the value of the FNP instruction counter at the time of the fault.

If the fault is "illegal opcode," there is usually an additional message of the form

module: message

where module is the name of the FNP program that detected the error, and message indicates the nature of the error.

A few of the FNP error messages indicate probable hardware problems, and should be referred to Honeywell Field Engineering if they occur frequently; these messages are listed below. Any other FNP messages indicate probable software problems.

unrecoverable i/o error
more than 5 consecutive i/o errors
3 consecutive mailbox checksum errors
receive transfer timing error
xmit transfer timing error
send transfer timing error

In addition, if the fault identified on the first line of the message is "memory parity" or "iom chan fault," there is probably an FNP hardware problem. In the latter case, the message giving the FNP instruction counter is replaced by a message of the form

channel NN, fault status = XXXXXX

where NN is a two-digit decimal number, and XXXXXX is a six-digit octal number.

The following FNP errors do not crash the system; all of them indicate possible hardware problems.

355 iom channel fault, channel NN, fault status XXXXXX
dia i/o error, status XXXXXX
abnormal lsla status XXXXXX
excessive hsla interrupts, line NN
trouble synchronizing lsla NN, some lines may not answer.

SECTION 2

COMMANDS

This section contains descriptions of those Multics commands (including the ones that extract the contents of the syserr log) that are useful for diagnosing system malfunctions. Each description contains the name of the command (with its abbreviation, if any), discusses its purpose, shows the correct usage, lists the arguments that can be used, and provides notes and, when necessary, examples.

Syntax lines give the order of required and optional arguments accepted by a command or active function. Optional portions of syntax are enclosed in braces ({}). The syntax of active functions is always shown enclosed in brackets ([]), which are required for active function use. To indicate that a command accepts more than one of a specific argument, an "s" is added to the argument name (e.g., paths, {paths}, {-control_args}).

Name: analyze_multics, azm

SYNTAX AS A COMMAND

```
azm {-control_args}
```

FUNCTION

invokes a subsystem that aids in system crash analysis. It can analyze dumps created by the BOS FDUMP command and copied into the Multics hierarchy by the copy_dump command.

CONTROL ARGUMENTS

-abbrev, -ab
: enables abbreviation expansion of request lines.

-no_abbrev, -nab
: does not enable abbreviation expansion of request lines. (Default)

-no_prompt
: suppresses the prompt for request lines in the request loop.

-no_start_up, -nsu
: does not execute any startup exec_com. (Default)

-profile PATH, -pf PATH
: specifies the pathname of the profile to use for abbreviation expansion. The suffix "profile" is added if necessary. This control argument implies -abbrev.

-prompt STR
: sets the request loop prompt to STR. The default is the ioa_ STR:
^/azm^[(^d)^]:^2x

-request STR, -rq STR
: executes STR as an azm request line before entering the request loop.

-start_up, -su
: executes the exec_com "start_up.azmec" upon invocation of azm. This start_up exec_com is first searched for in your home directory, then in your project directory (>udd>Project_id), and last in >site. The first exec_com found is used.

-quit
: exits azm after execution of other arguments. Can be used with -request.

NOTES

This command uses the standard search list mechanism to locate FDUMPs. If it does not find a "dumps" search list, it creates one, placing >dumps in the search list as the default. If additional search paths are desired, the add_search_path command can be used to define them.

VIRTUAL ADDRESS CONSTRUCTS

Accessing data requires some pointer value to define an address space. The generation of the pointer value is performed by resolving a virtual address (VIRTUAL-ADDR). A VIRTUAL-ADDR consist of two parts: a segment number and a word offset.

The command resolves VIRTUAL-ADDRs from the following types of information:

Symbol:
is a symbolic name for a segment number and an offset that can be resolved by data in definitions_ (i.e., sst$ptl can be resolved to the correct segment number and offset of the page table lock).

Segment name:
a segment name can be resolved in many ways, but it can only provide one part of the virtual address; azm uses 0 as the default offset for this pointer value (i.e., tc_data is resolved to SEGNO|0).

Segment number:
a segment number needs no resolution, but a default action needs to be taken for the offset (the default is 0, i.e., SEGNO|0).

Segment name/number and offset:
the VIRTUAL-ADDR in this case can be a segment name or segment number and an octal offset (i.e., the construct of pds|20 is translated to SEGNO|20 or dseg|5 is 0|5). The notation "|" and "$" must be used without spaces (e.g., 244|0 or sst$cmp).

Temporary pointers:
azm keeps a set of 11 temporary pointers per translation. A translation is one complete entity such as an "FDUMP". These pointers can be set with the set request (e.g., set sp 230|100). They can be referenced by other requests as another type of "symbol" in a VIRTUAL-ADDR expression, after they have been set. If not set, these pointers are null.

Offset operators:
the operators "+N" and "-N" immediately preceding an octal number, or VIRTUAL-ADDR construct can be used to alter the offset of a virtual address. N is a number interpreted in octal. No spaces are allowed between the operator and the N. For example, sst$ptl +30 are resolved to be the SEGNO for sst_seg with the offset of ptl plus 30 octal locations; sst$ptl+30 is also valid.

Indirection:

A VIRTUAL-ADDR can imply indirection. The indirect word can be used as an ITS pair if it is a valid ITS word pair; if not, the upper half of the word is used. The following VIRTUAL-ADDR construct is used to specify indirection (sst$cmp,*). The format of an indirect pointer value is

```
segno|offset,*  segname|offset,*  symbol,*
temp_ptr,*      temp_ptr|offset,*
```

EXAMPLES OF INDIRECTION

```
17|230,*  sst|230,*  sst$cmp,*+2
sp,*      sp|230,*
```

ANALYZE_MULTICS REQUESTS

absolute_address, absadr

SYNTAX

```
absadr VIRTUAL-ADDR
```

SYNTAX AS AN ACTIVE REQUEST

```
[absadr VIRTUAL-ADDR]
```

FUNCTION

translates a "virtual address" to an absolute memory address.

ARGUMENTS

VIRTUAL-ADDR
: can be a segment number, name, or symbolic address (e.g., 64, prds, prds$am_data). See "Virtual Address Constructs" above.

Active request example

```
! display_absolute [absadr sst$cmp] 2
```

displays the first two words of the absolute address of sst$cmp.

add_request_table, arqt

SYNTAX

```
arqt PATH
```

FUNCTION

adds a user-defined request table in the list of request tables being searched by the current azm invocation.

ARGUMENTS

PATH
is the pathname of the request table to be added. This request table must be consistent for use with the subsystem utility. (See Section 4 of the Programmer's Reference Manual for request table structure.)

apply, ap

SYNTAX

```
ap VIRTUAL-ADDR {RANGE} command_line
```

FUNCTION

extracts all or part of a segment, specified by VIRTUAL-ADDR from the selected FDUMP, and places a copy in a temporary segment. This pathname is passed as the last argument in the command_line.

ARGUMENTS

VIRTUAL-ADDR
may be a segment number, name or symbolic address (e.g., 64, prds, prds$am_data). See "Virtual Address Constructs" above.

RANGE
specifies the number of words in octal to be copied. The default is the entire segment.

command_line
is any command.

NOTES

The offset in the virtual address specifies where the copying of the segment begins. When only part of a segment is extracted, it goes at the beginning of the temporary segment. For example:

```
ap pds$am_data 400 dump_segment
```

puts 256 (decimal) words at the beginning of the segment.

apte

SYNTAX

```
apte {PROC_INDICATOR} {-control_args}
```

FUNCTION

displays active page table (apte) information for processes in an FDUMP that match the states specified.

ARGUMENTS

PROC_INDICATOR
used for specifying individual processes. It can take one of three forms:

- The decimal index (starting at zero) of a process in the FDUMP.
- The octal apte offset of the process.
- The octal process_id of the process.

CONTROL ARGUMENTS

-all, -a
displays apte information for all processes in any state. (Default)

-blocked, -blk
displays apte information for all processes in the blocked state.

-count, -ct
specifies the total number of processes meeting the criteria specified by control_args. With -all, it gives the counts of each process state.

-current, -cur
displays apte information for the current process.

-page_tbl_lock, -ptl
displays apte information for all processes marked as page table locking.

-ready, -rdy
displays apte information for all processes in the ready state.

-run
displays apte information for all processes in the running state.

-stopped, -stop
displays apte information for all processes in the stopped state.

-wait
displays active page table entry (apte) information for all processes in the waiting state.

EXAMPLES

```
apte 2
```

displays information for process 2 in the FDUMP.

```
apte 10600
```

displays information for the process with apte offset 10600 (octal).

```
apte 3500555555
```

displays information for the process with octal process_id 003500555555.

associative_memory, am

SYNTAX

```
am {-control_args}
```

FUNCTION

displays SDW and/or PTW associative memories.

LOCATION CONTROL ARGUMENTS

-dump
displays the "dump" associative memories from the BOS CPU at the time the dump was taken. (Default)

-prds
displays associative memories that have been stored in the prds of the processor on which the current process is running.

CONTROL ARGUMENTS

-all, -a
displays all entries in the associative memories. The default is to display only those entries that are valid (i.e., the full bit is on).

-ptw
displays only the PTW associative memories.

-pageno PAGENO
displays only those entries in the PTW associative memories that have a page number that matches the value of PAGENO (which is an octal page number).

-sdw
displays only the SDW associative memories.

-segno SEGNO
displays only those entries in the SDW and PTW associative memories that have a segment number that matches the value of SEGNO, which is an octal segment number. (See assoc_mem.incl.pl1.)

NOTES

If no control arguments are given, both the SDW and PTW associative memories are displayed for the "dump" associative memories.

aste

SYNTAX

```
aste segno/segname {-control_args}
```

FUNCTION

displays active segment table (ast), page table, and trailer information. The default displays active segment table entry (aste) and page table information only.

ARGUMENTS

segno/segname
is the segment number or segment name of interest.

CONTROL ARGUMENTS

-aste
displays active segment table information for the selected entry.

-at offset, -at virtual-addr
displays aste information starting at the offset or virtual address specified.

-brief, -bf
displays everything excluding the page table information.

-long, -lg
displays everything, that is, the aste, page table, and trailer information.

-page_table, -pt
displays page table information for the selected segment.

-trailer, -tr
displays trailer information about the selected segment.

configuration_deck, cd

SYNTAX

```
cd {card_names} {-control_args}
```

FUNCTION

displays the contents of the configuration deck in the selected FDUMP. This request works exactly like the standard pcd command, except that it gets the configuration deck from the FDUMP.

ARGUMENTS

card_names
are the names of the particular configuration cards to be displayed. Up to 32 card names can be specified (separated by spaces). If no card names are given, the the complete configuration deck is printed.

CONTROL ARGUMENTS

-brief, -bf
suppresses the error message when a requested card name is not found. (Default)

-exclude FIELD_SPECIFIERS, -ex FIELD_SPECIFIERS
excludes particular cards or card types from being displayed. One to 14 field specifiers can be supplied with each -exclude, and up to 16 -exclude control arguments can be specified. To be eligible for exclusion, a card must contain fields that match all field specifiers supplied with any -exclude argument.

-long, -lg
prints an error message when a requested card name is not found.

-match FIELD_SPECIFIERS
selects particular cards or card types to be displayed. One to 14 field specifiers can be supplied with each -match, and up to 16 -match control arguments can be specified. To be eligible for selection, a card must contain fields that match all field specifiers supplied with any -match argument.

NOTES

Field specifiers can consist of a complete card field or a partial field and an asterisk (*). An asterisk matches any part of any field. Specifiers for numeric fields can be given in octal or decimal, but if decimal they must contain a decimal point. Asterisks cannot be specified in numeric field specifiers. All numeric field specifiers are converted to decimal and matched against numeric card fields, which are also converted to decimal. Hence, the field specifier "1024." matches a card containing the octal field 2000, and the field specifier "1000" matches a card containing the decimal field 512. Note that all card names must be specified before the first -match or -exclude argument. Field specifiers following a -match or -exclude argument include all arguments until the next -match or -exclude argument.

display, d

SYNTAX

```
d VIRTUAL-ADDR {EXP} {RANGE} {-control_args}
```

SYNTAX AS AN ACTIVE REQUEST

```
[d VIRTUAL-ADDR {EXP} {RANGE} {-control_args}]
```

FUNCTION

displays a selected portion of a segment in the FDUMP..

ARGUMENTS

VIRTUAL-ADDR
: specifies the initial offset of the virtual address space to be dumped. May be a segment number, name, or symbolic address (e.g., 64, prds, prds$am_data). See "Virtual Address Constructs" above.

EXP
: is an expression, which is either an octal value or a VIRTUAL-ADDR construct yielding an octal value. This value can be positive or negative, specified by the plus or minus sign.

RANGE
: specifies the number of words to be dumped in octal. If a RANGE is not specified, the default action is to display one word. If the data is an ITS pair, two words are displayed.

MODE SPECIFICATIONS

-character, -ch, -ascii
: displays the selected number of characters in ASCII. Characters that cannot be printed are represented as periods. Usage as an active request is not allowed.

-instruction, -inst
displays the selected number of words as instructions. Usage as an active request is not allowed.

-octal, -oc
displays the selected number of characters in octal. When used as an active request, it returns the octal value of the requested address. (Default)

-ptr, -p
displays the selected number of word pairs as pointers. When used as an active request, it returns the octal value of the form SEGNO|OFFSET.

-pptr, -pp
displays the selected number of words as a packed pointer. When used as an active request, it returns the octal value of the form SEGNO|OFFSET.

-pptrx, -ppx
displays the selected number of words as packed pointers and expands the SEGNO|OFFSET to a segment name. Usage as an active request is not allowed.

-ptrx, -px
displays the selected number of word pairs as pointers and expands the SEGNO|OFFSET to a segment name. Usage as an active request is not allowed.

CONTROL ARGUMENTS

-as STRUCTURE_NAME
displays the data as a hardcore PL/I structure defined by STRUCTURE_NAME. The STRUCTURE_NAME is a hardcore system-defined include file. The address given in the display request is taken as the address of the beginning of the structure. If the whole structure is being displayed, that is the address where display begins. If only certain elements are being displayed, that is the address used to compute offsets of the elements. The structure reference following -as must be a single string, containing no spaces, and follows the syntax described below. The single string is used to specify structure elements, array indexes, and substring matching. Usage as an active request is not allowed.

-long, -lg
displays each element of the structure on a separate line. This control argument is only implemented with -as.

STRUCTURE SYNTAX

The structure reference is made up of two parts: a structure element reference and an optional set of match strings. If no match strings are supplied, no string matching is done. The structure element reference syntax consists of one or more element names, separated by periods, and may contain subscripts following some of these element names. The first name in a structure element reference must be a level-one structure reference; partially qualified top-level references are not permitted. Intermediate levels of qualification may be omitted as long as there is no ambiguity.

All subscripts must be supplied as decimal integers. The subscripts can be cross-section references such as "(1:4)" to reference elements one through four. Asterisk bounds cannot be used: if a cross section is desired, its upper and lower bounds must be given as decimal constants. If an element has more subscripts than are supplied, the complete cross section is printed for the remaining subscripts. To eliminate the need for quoting, subscripts may be surrounded by braces instead of parentheses.

In order to specify that only certain elements be displayed (such as all those with names containing the string "time"), a set of match strings can be given after the structure element reference. Each match string begins with a slash and is followed by the string itself. The final match string can be followed by a slash, but this is not required. If match strings are specified, any element that matches at least one string is displayed.

EXAMPLES OF STRUCTURE REFERENCES

pvt
the whole structure "pvt".

pvt.n_entries
the single element "n_entries" in the structure "pvt".

sst/time/, sst/time
any elements in the structure "sst" containing the string "time". Note that the final slash is optional.

sst/time/meter/
any elements in the structure "sst" containing either the string "time" or the string "meter".

sst.space{3}
element three of "sst.space".

sst.space{2:4}
elements two, three, and four of "sst.space".

sst.space
all elements of "sst.space".

sst.level{1}
both elements of the "level" array for "sst.level{1}"

sst.level{1}.ausedp, sst.level.ausedp{1}
the single element "ausedp" of the "level" array for "sst.level{1}"

STRUCTURE OUTPUT FORMAT

The default output format is a compressed form, which places as many values on a line as will fit within the line length. The -long control argument places one value on a line. The short form, additionally, collects all bit(1) flags and displays them, at the end of the display for each substructure or array element, in two groups: one listing all the flags that were on ("1"b) and one for all the ones that were off ("0"b).

All PL/I data types are displayed in the same representations used by probe. Additionally, the following special formats are used:

1. Bit strings are displayed in octal if the length is divisible by three, in hex if divisible by four, and as bit strings otherwise.

2. Character strings are displayed as a string concatenated with a repeated constant if the string is padded on the right with more than 16 nulls, spaces, or octal 777 characters.

3. Large-precision (> 51) fixed binary values are also displayed as clock readings if their values represent clock readings within 10 years of the present.

EXAMPLES

```
d 75|560 2
```

displays the two words in seg number 75 starting at offset 560.

```
d pds|560 2
```

displays the two words in the segment named pds starting at offset 560.

```
d pds$trace
```

displays one word in the pds segment beginning at the offset specified by $trace.

```
display 244|260 +20 4
```

displays four words of segment number 244 starting at offset 300 octal.

```
d sp 20
```

displays 20 octal words starting with the segment offset defined in the azm internal temporary pointer (see set request).

```
d sst$cmp,* +sst$cmesize sst$strsize
```

causes the word at sst$cmp to be used as an indirect word, or an indirect pointer if the resultant address has ITS modification, to develop the starting virtual address. The value derived from sst$cmesize is then added to the starting offset for the "final" starting address. The range, or number of words to be displayed, is specified by the value contained in sst$strsize.

```
d sst|2 -as apte
```

displays the APTE entry at the given offset in the SST as it is defined by apte.incl.pl1.

display_absolute, da

SYNTAX

```
da ABS-ADDR {RANGE} {-control_args}
```

SYNTAX AS AN ACTIVE REQUEST

```
[da ABS-ADDR {RANGE} {-control_args}]
```

FUNCTION

dumps an absolute memory address space in the FDUMP.

ARGUMENTS

ABS-ADDR
is the starting absolute memory address, in octal.

RANGE
specifies the number of words to be dumped in octal. If a range is not specified, the default is one word. If the data to be dumped is an ITS pair, two words are dumped.

MODE SPECIFICATIONS

For a description of the mode specifications, see the display request.

-character, -ch, -ascii
-instruction, -inst
-octal, -oc
-ptr, -p
-pptr, -pp
-pptrx, -ppx
-ptrx, -px

events, ev

SYNTAX

```
ev {-control_args}
```

FUNCTION

displays significant events, in reverse chronological order, from an FDUMP (see "Notes").

CONTROL ARGUMENTS

-last N, -lt N
specifies the number of events to print. The default is to print all.

-long, -lg
displays disk queue events.

-time NSECS, -tm NSECS
specifies the time in seconds before the dump was taken when events were significant. The default is 10 seconds.

NOTES

The following events are considered as significant: machine conditions (from BOS, prds, pds, and the mc_trace_buf), traffic control state change time, Syserr messages (from both syserr_data and syserr_log), Fim frames in any stack, and connects by device and disk queues (long report only).

history_regs, hregs

SYNTAX

```
hregs {HREGS_specifier} {-control_args}
```

FUNCTION

displays a composite analysis or octal dump of the processor history registers. This request is useful for people who are knowledgable of the hardware. The default action is to display the AU, CU, DU, and OU history registers for the pds in a threaded order and interpreted format.

HREGS SPECIFIERS

-condition VIRTUAL-ADDR, -cond VIRTUAL-ADDR
displays history registers from a condition frame, the location of which is described by VIRTUAL-ADDR.

-dump
displays the "dump" history registers from the BOS CPU at the time the dump was taken.

-pds
displays the history registers that have been stored in the current processes pds. (Default)

VIRTUAL-ADDR
displays the history registers that have been stored at the address space specified by VIRTUAL-ADDR.

CONTROL ARGUMENTS

-au
displays the AU history registers only.

-cu
displays the CU history registers only.

-du
displays the DU history registers only.

-ou
specifies that only the OU history registers are to be displayed.

-interpret
displays the interpreted form of the history registers only (default), or, if -octal is given, include the octal representation also.

-octal, -oc
displays the octal values of history registers only, or, if -interpret is also selected, display octal and interpreted form. If neither -octal nor -interpret is specified, the default action is to display the interpreted form only.

-thread
displays the selected history registers in the "correct" order. (Default)

-no_thread
display the selected history registers in serial order, without attempting to sort them.

list_dumps, lsd

SYNTAX

```
lsd {PATH}
```

FUNCTION

lists the FDUMPs in the selected dump directory. If PATH is not given, FDUMPs from all the dump directories specified in the dumps search list are listed.

ARGUMENTS

PATH
is the pathname of the dump directory to be listed.

list_processes, lsp

SYNTAX

```
lsp {PROC_INDICATOR} {-control_args}
```

SYNTAX AS AN ACTIVE REQUEST

```
[lsp {PROC_INDICATOR} {-control_args}]
```

FUNCTION

lists all known processes in the selected FDUMP. As an active request, it returns the process_ids meeting the control argument criteria.

ARGUMENTS

PROC_INDICATOR
used for specifying individual processes. It can take one of three forms:

- The decimal index (starting at zero) of a process in the FDUMP.
- The octal apte offset of the process.
- The octal process_id of the process.

CONTROL ARGUMENTS

-all, -a
lists all processes in the FDUMP. (Default)

-blocked, -blk
lists processes marked as blocked.

-count, -ct
counts all processes. With -all, it gives the counts of each process state.

-current, -cur
lists the current process.

-page_tbl_lock, -ptl
lists processes marked as page table locking.

-ready, -rdy
lists processes marked as ready.

-run
lists processes marked as running.

-stopped, -stop
lists processes marked as stopped.

-wait
lists processes marked as waiting.

EXAMPLES

```
! do "select_process &1;sdw 0" ([list_processes])
```

displays the SDW for DSEG for all processes in the FDUMP.

machine_conditions, mc

SYNTAX

```
mc {MC_specifier} {-control_args}
```

FUNCTION

displays all or parts of machine conditions based on the given pointer.

MC SPECIFIERS

-dump
specifies the dump for the BOS CPU regs at time of dump.

-pds {STR1}
where STR1 can be "all", "fim", "page_fault" ("pgf"), "signaller" ("signal", "sig"). It defaults to "all" if STR1 is not given.

-prds {STR2}
where STR2 can be "all", "fim", "interrupt" ("int"), "system_trouble" ("sys"). It defaults to "all" if not given.

VIRTUAL-ADDR
is the virtual address construct used to define the address space containing machine conditions.

CONTROL ARGUMENTS

-eis
displays the EIS pointers and lengths (interpreted).

-faults, -flt
displays the fault register.

-long, -lg
displays all elements of the MC.

-mc_err
displays the mc_err data word.

-misc
displays the miscellaneous data (i.e., mc_err, fault reg, time).

-octal, -oc
displays the eis info, scu data, or pointer registers, in octal. This control argument is used with -scu, -eis, or -regs.

-pointers {PR_LIST}, -prs {PR_LIST}
displays pointer registers selected by PR_LIST (from 0 to 7, separated by spaces). If PR_LIST is not specified, all the pointers are displayed.

-ppr
displays only the PSR and IC from the MC.

-registers {REG_LIST}, -regs {REG_LIST}
displays only the basic OU registers. Where REGS_LIST can be any of the following:

x0 x1 x2 x3 x4 x5 x6 x7 a q all.

If REG_LIST is not specified, all of the basic OU registers are displayed.

-scu
displays only the scu data of the MC.

-time, -tm
displays the MC time.

-tpr
only displays the TSR and the CA from the MC.

NOTES

If no MC specifiers are given, the temporary pointer prmc is used. The default control arguments are -eis, -fault, -mc_err, -pointers, -scu, -time, and -tpr. Either -pds or -prds must be supplied. The machine_conditions request sets all azm-defined temporary pointers as seen in the machine_condition frame.

EXAMPLES

```
mc -pds fim -scu
```

displays the scu data found in the fim frame of the pds currently being referenced in the dump.

page_trace, pgt

SYNTAX

```
pgt {-control_arg}
```

FUNCTION

displays the contents of the page trace table in the current process data segment (PDS). The default is to display the last 15 trace entries. Trace entries are always displayed in reverse chronological order.

CONTROL ARGUMENTS

-all, -a
displays all trace entries.

-last N, -lt N
specifies the number of trace entries, where N is a positive decimal integer, to be displayed.

replace, rp

SYNTAX

```
rp segno/segname PATH
```

FUNCTION

replaces the segment designated by segno/segname in the current translation table, with another segment designated by PATH.

ARGUMENTS

PATH
is the pathname of the segment. The equal convention can be used: rp bound_system_faults [e wd] >=.new

segno/segname
the segment number or segment name within the translation table to be replaced.

NOTES

Both per-process and per-system segments can be replaced. For example, if the pds is replaced in a process, it affects only the current process; whereas if tc_data is replaced in a process, it affects the whole FDUMP.

scus

SYNTAX

```
scus
```

FUNCTION

prints the memory address space (in octal) of each scu from the registers saved in the FDUMP.

sdw

SYNTAX

```
sdw {segno/name} {segno/name}
```

FUNCTION

displays the SDWs in the current processes DSEG.

ARGUMENTS

segno/name
is the segment number or name of interest. The first is the starting segment number and the second is the ending segment number. If only one is given, only one is displayed; if none are given, all are displayed.

search, srh

SYNTAX

```
srh VIRTUAL-ADDR {RANGE} SEARCH_STRING
```

SYNTAX AS AN ACTIVE REQUEST

```
[srh VIRTUAL-ADDR {RANGE} SEARCH_STRING]
```

FUNCTION

searches a segment starting at VIRTUAL-ADDR matching on SEARCH_STRING. The search is performed on a 36-bit-word boundary. As an active request, the virtual addresses matching the criteria specified is returned.

ARGUMENTS

VIRTUAL-ADDR
is the pointer to the address space to search.

RANGE
specifies the number of words to be searched from the starting offset, where range is an octal value. The default is the rest of segment. The search is started from VIRTUAL-ADDR.

SEARCH_STRING
is a 12-character string representing the 12 octal digits that make up a machine word (36 bits, 3 bits per digit). This forms both the search data and search mask by using the hyphen (-) as a "don't care character" in the string. The "do care digits" are octal "from 0 to 7." Any other character is illegal.

EXAMPLES

To search for

1. all words in segment 76 that have the last two digits of 43:

```
srh 76 ----------43
```

2. all words in tc_data where the upper half = 070707:

```
srh tc_data 070707------
```

3. words that end in 1234 in sst_seg starting at 1000 but only searching for 200 octal words:

```
srh sst_seg|1000 200 --------1234
```

4. words that start with 45 and end with 77 starting a sst_seg$ptl for 100 words:

```
srh sst_seg$ptl 100 45--------77
```

segment_name, name

SYNTAX

```
name VIRTUAL-ADDR

    name number
```

SYNTAX AS AN ACTIVE REQUEST

```
[name VIRTUAL-ADDR]

    [name number]
```

FUNCTION

prints the segment name given a virtual address or a segment number. .

ARGUMENTS

VIRTUAL-ADDR
is the virtual address construct used to define the segment.

number
is the segment number of the segment to be referenced. Thus, "name 230" returns the name associated with the segment number 230, which is "stack_0".

segment_number, number

SYNTAX

```
number VIRTUAL-ADDR

    number name
```

SYNTAX AS AN ACTIVE REQUEST

```
[number VIRTUAL-ADDR]

    [number name]
```

FUNCTION

prints the segment number given either a virtual address or a segment name.

ARGUMENTS

VIRTUAL-ADDR
is the virtual address construct used to define the segment.

name
is the name of a segment, e.g., stack_0. Thus, "number sst_seg" returns the segment number associated with the segment sst_seg, which is "77".

select_dump, sld

SYNTAX

```
sld {NAME} {-control_args}
```

FUNCTION

selects and translates an FDUMP of a system crash. Found via the dump search list, which defaults to >dumps.

ARGUMENTS

NAME
is the ERF number or the path name of the zero component of the FDUMP. It can also be the form path>35, where 35 is the erf number. Several control arguments are also acceptable if NAME is not specified.

CONTROL ARGUMENTS

-first, -ft
selects the first dump (by erf number) in the dump directory found via the dump search list.

-last, -lt
selects the last (most current) dump in the dump directory according to erf number.

-next, -nx
selects the next dump in the dump directory. This is relative to the dump currently being looked at.

-previous, -prev
selects the previous dump in the dump directory. This is relative to the dump currently being looked at.

select__process, slp

SYNTAX

```
slp {PROC_INDICATOR} {-control_args}
```

FUNCTION

selects a process for examination. When invoked with no arguments, the current process is listed.

ARGUMENTS

PROC_INDICATOR
used for specifying individual processes. It can take one of three forms:

- The decimal index (starting at zero) of a process in the FDUMP.
- The octal apte offset of the process.
- The octal process_id of the process.

CONTROL ARGUMENTS

-brief, -bf
suppresses the message about changing processes.

-cpu TAG
selects the DBR for the process running on the CPU identified by TAG (where TAG is one character in the range a through h).

-dbr dbr_value
selects the process defined by the dbr_value.

-long, -lg
prints a message announcing the new process selected. (Default)

set

SYNTAX

```
set PTR_N VIRTUAL-ADDR
```

FUNCTION

sets an internal temporary pointer like a cpu pointer register (i.e., "pr6" or "sp"). These pointers can then be used as a VIRTUAL-ADDR by other azm requests.

ARGUMENTS

VIRTUAL-ADDR
can be a segment number, name, or symbolic address (e.g., 64, prds, prds$am_data).

PTR_N
can be either the name or number of a "temporary pointer."

There are eight temporary pointers and two special-case pointers.

NUMBER	NAME	NUMBER	NAME
pr0	ap	pr4	lp
pr1	ab	pr5	lb
pr2	bp	pr6	sp
pr3	bb	pr7	sb

prmc	intended to be a pointer to the current MCs.
prfr	intended to be a pointer to the current stack frame.

EXAMPLES

```
set pr6 240|100
```

this sets a temporary ptr named pr6 (sp).

```
set sb 240
```

this sets the temporary ptr (sb) to the base of seg 240 (240|0).

NOTES

The value of a temporary pointer can be displayed via the value request: v {ptrn | -all}

stack, sk

SYNTAX

```
sk VIRTUAL-ADDR {-control_arguments}
```

FUNCTION

traces a given stack.

ARGUMENTS

VIRTUAL-ADDR
is the virtual address construct defining the stack to be traced.

CONTROL ARGUMENTS

-arguments, -ag
prints the arguments for the stack frames traced.

-for N
traces for N stack frames. If no valid stack frames exist (stack_begin_ptr = stack_end_ptr), a -force must be used.

-force, -fc
forces a forward stack trace. For use when there are no valid frames for this stack (stack_begin_ptr = stack_end_ptr).

-forward, -fwd
traces in a forward manner.

-long, -lg
prints the arguments and an octal dump of the stack frames traced.

NOTES

The default is to trace the stack in reverse order unless -force or -forward are specified. If the VIRTUAL-ADDR has a zero offset, then the trace starts at the offset of the first stack (stack_header.stack_begin_ptr). If it has a nonzero offset, then the trace is started from that offset in the given stack.

syserr_log, slog

SYNTAX

```
slog {-control_args}
```

FUNCTION

displays all or parts of the syserr_log and syserr_data segments from the dump. It does not examine the perm_syserr_log. The default is to print the entire log.

CONTROL ARGUMENTS

-action A
displays only messages with an action code specified by A, where A is a decimal integer in the range 0 to 9.

-exclude STR -ex STR
excludes any message that contains STR, where STR is a string that is matched against messages in the log.

-last N, -lt N
starts the scan N messages back from the end of the log, where N is a decimal integer.

-match STR
displays any message that contains STR, where STR is a string to be matched against messages in the log.

-expand, -exp
interprets the binary data of messages. The format is generally dependent on the text of the message.

traffic_control_queue, tcq

SYNTAX

```
tcq {-control_args}
```

FUNCTION

displays process DBR, process state, process ID, current CPU, and user ID from the Traffic Controller's Eligible Queue, as well as the "process number" in the FDUMP. The default is to display only the eligible queue.

CONTROL ARGUMENTS

-all
displays the eligible, real-time, interactive, and work-class queue entries, including the unthreaded entries.

-ready, -rdy
displays the eligible, real-time, interactive, and work-class queues, excluding the unthreaded entries.

value, v

SYNTAX

```
v PTR_Ni...PTR_Nn

    v -all
```

FUNCTION

displays the current value of one or all of the temporary pointers.

ARGUMENTS

PTR_N
specifies which of the temporary pointers is to be displayed. Refer to the set request for a list of the azm-defined pointer names.

-all, -a
specifies that all of the pointers are to be displayed. (Default)

verify_associative_memory, vfam

SYNTAX

```
vfam {-control_args}
```

SYNTAX AS AN ACTIVE REQUEST

```
[vfam {-control_args}]
```

FUNCTION

performs a consistency check on the associative memories stored at the time of a dump by comparing them to the appropriate entries in the "dump dseg" and page tables. When used as an active request, returns "true" if any inconsistencies are found, "false" otherwise.

CONTROL ARGUMENTS

-ptw
restricts the verification to the PTW associative memories.

-sdw
restricts the verification to the SDW associative memories.

NOTES

If no argument is given, both SDW and PTW associative memories are checked.

why

SYNTAX

why

FUNCTION

tries to find the stack that has a call to syserr_real$syserr_real or call_bos$call_bos and sets the temporary pointers pr6 and prfr to the stack frame. This request searches the stacks for a frame that has a return_to_ring_0_ frame and sets the temporary pointers from this set of machine conditions that called this entry.

NOTES

If the crash is due to fim_util$check_fault finding a problem, the machine condition CU data is displayed and all temporary pointers are set from these machine conditions. If this is an execute fault, then some lock info is printed and the process selected is locked (look at PTL first then ASTL).

If this fdump is due to a manual return to BOS, then some pertinent lock info is also printed.

STANDARD SUBSYSTEM REQUESTS

.
prints a line describing the current invocation of azm.

?
prints a list of requests available in azm.

abbrev, ab
controls abbreviation processing of requests lines.

answer
provides preset answers to questions asked by another request.

do
executes/returns a request line with argument substitution

execute, e
executes a Multics command line.

exec_com, ec
executes a file of azm requests that can return a value.

help
prints information about azm requests and other topics.

if
conditionally executes/returns one of two request lines.

list_help, lh
displays the name of all azm info segments on given topics.

list_requests, lr
prints a brief description of selected azm requests.

quit, q
exits azm.

ready, rdy
prints a Multics ready message.

ready_off, rdf
disables printing of a ready message after each request line.

ready_on, rdn
enables printing of a ready message after each request line.

subsystem_name
prints/returns the name of this subsystem.

subsystem_version
prints/returns the version number of this subsystem.

The standard escape convention for executing Multics command lines (..) is also supported.

Name: check_cpu_speed

SYNTAX AS A COMMAND

```
check_cpu_speed {cpu_tags}
```

FUNCTION

performs a relative check of the speed of a currently running CPU on the system.

ARGUMENTS

cpu_tags
are the tags of CPUs configured on the system. If more than one is supplied, the values must be separated by spaces. The default is to run on all CPUs listed by the list_proc_required command that are currently marked as ON in the configuration deck.

ACCESS REQUIRED

This command requires access to the phcs_ gate to run.

NOTES

Your process is left running with the original set of system CPUs.

The command runs on a CPU outside of your original set of CPUs if the CPU tag is given on the command line.

Name: daily_syserr_process

SYNTAX AS A COMMAND

```
daily_syserr_process {-control_arg}n
```

FUNCTION

runs once a day to process the syserr log for the preceding day. It writes out all syserr log entries on various I/O switches.

CONTROL ARGUMENTS

-from DT
where DT is a date/time acceptable to convert_date_to_binary_ (see the Subroutines manual).

If the control argument is given, the syserr log is scanned for the first line written after the date specified. Log processing starts from this entry. If it is not given, the program looks for the sys_admin_data segment in the current working directory and uses the log_control structure there to look for information describing the index of the last entry previously processed.

Distribution of the output is controlled by the control file, syserr_select_file, in the current working directory. This file has comment lines beginning with an asterisk (*), and selector lines of the form

switch_name,S,opcode,text

where:

1. switch_name
   is the name of the I/O switch on which a line is written.

2. S
   is the syserr action code to be considered. See Section 1 for an explanation of these codes. S can also be specified as "*" to indicate all action codes.

3. opcode
   is the operation code. Valid opcodes are

   all
   selects all lines (at this syserr code)

   any
   selects all lines containing text

   begin
   selects all lines beginning with text

   not
   inhibits all lines not containing text

   nbegin
   inhibits all lines not beginning with text

   count
   counts all lines containing text

   bcount
   counts all lines beginning with text

   allx
   like all, but messages with binary data have the data expanded when printed

anyx
like any, but messages with binary data have the data expanded when printed

beginx
like begin, but messages with binary data have the data expanded when printed

4. text
is optional text that is the operand of opcode.

The opcodes not and nbegin must precede any selector lines they are to inhibit for a given I/O switch.

Each entry in the syserr log is considered. For each entry, the selector lines in syserr_select_file may or may not select the entry, depending on the operation code. If the entry is selected, it is written on the named switch. A syserr log entry can are selected (and therefore written) more than once if different switches are named.

All I/O switches named must be attached and open before daily_syserr_process is called.

At the end of processing, total lines are written. Then, if any lines were selected, a total count is printed out.

NOTES

This command is intended to be used by the "crank" absentee job run by the system administrator every day. This job is controlled by the contents of master.ec contained in >udd>SysAdmin>lib. The daily_syserr_process command itself is controlled by the syserr_select_file segment contained in >udd>SysAdmin>accounting_library.

The system administrator and site analysts should be consulted whenever these files must be modified so that they are more suitable for an individual installation.

EXAMPLES

Included here are excerpts of the files that are used at a representative Multics site. These files cause all hardware-oriented messages for a given day to be printed in the segment daily_log_1.

daily_syserr_process

syserr_select_file

```
* syserr log control file.
*
daily_log_0,5,bcount,ioi_interrupt: I/O error.
daily_log_0,5,nbegin,ioi_interrupt: I/O error.
daily_log_0,*,all
daily_log_0,3,bcount,RCP: Mount Reel
daily_log_1,3,bcount,RCP: Mount Reel
daily_log_1,1,all
daily_log_1,2,all
daily_log_1,0,bcount,op-not-complete
daily_log_1,0,bcount,parity
daily_log_1,3,bcount,pxss: notify time out                                    *
daily_log_1,0,bcount,on_line_salvager: begin salvaging
daily_log_1,0,bcount,ITT overflow
daily_log_1,0,begin,Now terminating user process
daily_log_2,3,nbegin,RCP: Mount
daily_log_2,3,not,setting timax
daily_log_2,0,nbegin,RCP: Detaching
daily_log_2,0,nbegin,RCP: Force Detaching
daily_log_2,5,bcount,ioi_interrupt: I/O error.
daily_log_2,5,nbegin,ioi_interrupt: I/O error.
daily_log_2,*,all
network_log,*,any,imp
network_log,*,any,network
*
* END
```

The following excerpt from master.ec shows how the daily log files are attached and opened. Then daily_log_process is invoked to fill the files. The files are then closed, detached, and dprinted.

master.ec

```
.
.
.
io_call attach daily_log_0 vfile_ daily_log_0
io_call attach daily_log_1 vfile_ daily_log_1
io_call attach daily_log_2 vfile_ daily_log_2
io_call attach network_log vfile_ network_log
io_call open daily_log_0 stream_output
io_call open daily_log_1 stream_output
io_call open daily_log_2 stream_output
io_call open network_log stream_output
daily_syserr_process
io_call close (daily_log_0 daily_log_1 daily_log_2 network_log)
io_call detach (daily_log_0 daily_log_1 daily_log_2 network_log)
.
.
.
exec_com util dp (daily_log_2 bwchart.print) assurance1 2
exec_com util dp (sumry cutrpt daily_log_0 crank.absout) accts0 2
exec_com util dp (daily_log_1 bwchart.print) admin0 2
exec_com util dp daily_log_2 sysprg2
exec_com util dp (daily_log_0 bwchart.print sumry crank.absout) admin1 2
.
.
.
.
```

Name: device_meters, dvm

SYNTAX AS A COMMAND

```
device_meters {-control_args}
```

FUNCTION

* prints out metering information for the page control devices. Information can be printed for the disk subsystems.

CONTROL ARGUMENTS

-error, -er
prints out error occurrence statistics for the device (see Notes below).

-reset, -rs
resets the metering interval to begin with the last call with -reset specified. If -reset has never been given in a process, it is equivalent to having been specified at system initialization time.

-left
prints out information regarding available space on each device.

-io
prints I/O volume statistics for each device.

-latency, -lat
prints device latency (delay) statistics.

-report_reset, -rr
generates a full report and then performs the reset operation.

NOTES

If no control argument is given, a full report is generated.

The following items are printed if -error is specified:

EDAC Corr. Errs
is a count of the times a read was performed and an error occurred, but the EDAC (error-detection-and-correction) hardware was able to correct the error.

Recov. Errors
is a count of the times an error occurred and the EDAC hardware was unable to correct it, but a subsequent retry resulted in proper transmittal of the data.

Fatal Errors
is a count of the occurrences of nonrecoverable errors.

Name: display_cpu_error

SYNTAX AS A COMMAND

```
display_cpu_error {-control_args}
```

FUNCTION

scans the syserr log and displays machine conditions and history registers.

CONTROL ARGUMENTS

-from DT, -fm DT
starts scanning the log at the date/time given.

-to DT
stops scanning the log at the date/time given.

-for T
computes the ending time from the starting time, where T is a relative time (such as "1hour").

-cpu CPU_LIST
displays information for the CPUs specified, where CPU_LIST is a list of CPU tags ("a c").

-nothread
specifies that the history registers are not to be threaded. The history registers will be output in octal with no interpretation. The default is off.

-expand, -exp
specifies that the history registers are not to be threaded but that they are to be interpreted.

NOTES

If -from DT is not given, the scan starts with the earlies entry in the syserr log. The ending time can be specified by using -for or -to, but not both. If both are omitted, the scan terminates with the last entry in the log. All dates and times must be in a format acceptable to convert_date_to_binary_ (see the Subroutines manual).

You must have re access to audit_gate_ and r access to the perm_syserr_log to use this command.

Name: display_syserr_log_part

SYNTAX AS A COMMAND

```
display_syserr_log_part {args}
```

FUNCTION

displays portions of the syserr logging partition that exist on the disk, in order to diagnose and correct problems that might occur in the syserr logging partition.

ARGUMENTS

header, he
prints the syserr log partition header.

check D, ck D
checks message threads in direction specified by D and validates message formats. Direction is specified by one of the following:

forward, f	forward checking only
reverse, r	reverse checking only

message N, msg N
displays a single message with message header information, message text and expanded binary output, and octal message words. The message to be displayed is specified by appending a positive or negative integer to the message argument:

N	displays the log message that is N from the top
-N	displays the log message that is N from the last

CONTROL ARGUMENTS

-offset ADDR, -ofs ADDR
displays message at word offset ADDR from the beginning of the syserr log.

-number N, -nb N
displays the syserr log message whose message number is N (decimal).

NOTES

If no arguments are specified, all logging partition information is displayed. You must have re access to >system_library_1>phcs_ in order to use this command.

Name: dump_firmware

SYNTAX AS A COMMAND

```
dump_firmware path mem {addr count}
```

FUNCTION

is used to dump the contents of a segment containing MPC firmware.

ARGUMENTS

path
: is the pathname of the segment containing the firmware.

mem
: must be "cs" to dump the control store overlay, "rw" to dump the read/write overlay, or "size" to print the locations and lengths of overlays in the module. If this argument is "size," no further arguments need be given; otherwise, the addr and count arguments described below must be given.

addr
: is the starting address to dump, in hexadecimal. This argument must be given if the mem argument is not "size."

count
: is the number of words to dump, in hexadecimal. This argument must be given if the mem argument is not "size."

Name: dump_mpc

SYNTAX AS A COMMAND

```
dump_mpc mpc_name {-control_args}
```

FUNCTION

performs a dump of the read/write memory of a MPC and selectively edits the dump, the trace table, and MPC and device statistics.

ARGUMENTS

mpc_name
: is the name of the MPC to be dumped. This name must appear on an MPC card in the config deck. If this argument is omitted, -channel must be given.

CONTROL ARGUMENTS

-dump
displays a hexadecimal dump.

-trace
displays an interpreted trace of the MPC.

-extend, -ext
extends the output file if it exists. The default is to overwrite the file.

-stat
displays the MPC and device statistics.

-mpc
displays MPC error data only.

-channel channel_name, -chn channel_name
specifies a channel name, where channel_name is of the form [iomtag] [channel_no] (i.e., a14). The iomtag field must be a tag of a configured IOM and the channel_no must be a decimal channel number. If this control argument is used, the mpc_name argument is optional. If both are used, the channel must be connected to the mpc specified.

-output_file {path}, -of {path}
directs dump output to the segment specified by path. If path is not given, a default segment name of [mpc_name].list is used. If this control argument is not given, the default is to direct output to your terminal.

-long
formats output for devices with 132 columns or more. The default is based on output type and can be used to override the file output default.

-short
formats output for devices with fewer than 132 colums. The default is based on output file type and can be used to override the file output default.

NOTES

If neither the -stat, -dump, -mpc, nor -trace control arguments are specified, only the MPC and device statistics are displayed.

Switch 4 on the MPC maintenance panel is used to control tracing in the MPC. Tracing is only done if this switch is in the down position. If the trace table is being dumped to see the events leading up to a particular error condition, it may be useful to place switch 4 in the up position as soon as possible after the error occurs. This inhibits further tracing of I/O in the MPC and reduces the chances of losing trace data caused by the table wrapping around before the dump can be taken.

The dump produced by this command is in a format similar to the BOS MPCD command described in the MOH.

*
You must have re access to rcp_priv_ to use the dump_mpc command.

Name: eis_tester, et

SYNTAX AS A COMMAND

```
eis_tester path {-control_args}
```

FUNCTION

sets up and tests EIS instructions in a controlled environment. You must prepare an input script describing the EIS instructions to be tested. From this input script the EIS tester builds the EIS instructions (one at a time) and the indirect words, descriptors, and data that each instruction needs. The instruction to be tested is set up in a special ALM segment (etx). The eis_tester command calls etx in order to execute the EIS instruction; etx returns to eis_tester when the instruction has been executed. After executing the instruction, eis_tester tests correct execution of the instruction. If one of the test scripts in the ets data base fails and the successful execution of that test is dependent upon installation of a particular FCO, the FCO number is displayed in the error message.

ARGUMENTS

path
is the pathname of a segment that contains input script data that defines the instructions to test.

CONTROL ARGUMENTS

-brief, -bf
suppresses all output except identification and error messages.

-nox
sets up the instruction but does not execute it; used to test the validity of the input script.

-debug
runs the test in a debugging loop where each instruction is tested 10 times but results from the test are not checked. Each time through the loop the instruction is set up completely, including all the specified faults.

-select N, -sel N, -do N
processes only test N (where N is a positive decimal number). This number has no relationship to the -ns field in any test.

-help
displays a brief usage statement.

-instruction_type INSTR, -inst INSTR
processes only tests that contain the instruction INSTR.

-long, -lg
displays all the related test information prior to executing a test.

-repeat N, -rpt N
repeats the entire execution of the selected tests N times.

-stop_on_failure, -sof
displays the failing data, machine condition, and history register information and return to command level if an error is detected in a test. The default is to display the failing data and continue with the next test.

-from N, -fm N
starts processing test N (where N is a positive decimal number) and continues processing all remaining tests in the input segment unless -to is used.

-to N
stops processing after test N (where N is a positive decimal number). If -from is not used, tests one through N are processed.

NOTES

Before eis_tester calls etx to execute the instruction, it sets up some special padding around the data field that is modified by the EIS instruction. Eight special characters (octal 717) are put in front of and at the end of the result data string. The result area itself is initialized to all zero bits.

When called to execute the EIS instruction, etx does the following:

1. Saves the current pointers and registers.

2. Loads the pointers and registers from values set up by eis_tester. These are the values of the pointers and registers when the EIS instruction is actually being executed.

3. Sets indicators to preinstruction test values.

   a. All indicators except b and c below are off.

   b. The BAR MODE indicator is always on.

   c. If the test instruction is expected to turn on any of the three overflow indicators (ov, eo, eu), then the om (overflow mask) indicator is turned on so an overflow fault is not taken.

4. Transfers to the instruction area in etx itself where eis_tester has set up the EIS instruction and its descriptors.

5. After the EIS instruction has been executed, etx stores the values of the indicators, pointers, and registers so that eis_tester can examine them.

6. Reloads the pointers and registers that were saved by etx.

7. Returns to eis_tester.

After the execution of the EIS instruction, eis_tester makes the following tests:

1. Checks to see that the data resulting from the instruction is correct.

2. Checks to see that the indicators are set correctly.

3. Checks to see that a truncation fault was correctly taken or not taken.

Instruction Area

The EIS instruction is set up in a special area in etx. This area consists of seven words. The first three words of the instruction area are set up in the last three words of a page. The last four words of the instruction area are the first four words of the next page. By positioning the instruction in the instruction area, you can position the instruction on a page boundary. Those words in the instruction area that are not used for the EIS instruction itself are set up as nop instructions. The default position of the instruction word is in instruction area word 4. This places the instruction at word 0 of a page.

```
PAGE  A          WORD 1
                 WORD 2
                 WORD 3
              ---------------
PAGE  B          WORD 4          <--  Default position of
                 WORD 5               instruction word.
                 WORD 6
                 WORD 7
```

Data Areas

The data referenced by each descriptor of the instruction is set up in a special data area. There is one data area used for every descriptor of the instruction. Each data area consists of three pages. The default starting position of the data is character 0 of word 0 of page 2. The last 32 words of page 1 and the first 32 words of page 3 can also be used to hold the test data. Thus the maximum data size of any string is 1088 words or 4352 (9-bit) characters. You can position the start of the data string so that it starts in page 1. Thus you can define data strings that cross page boundaries. A long data string can cross two page boundaries.

Notes

Depending on what modification is used by the instruction, the data areas used may or may not be in the same segment.

If a descriptor is referenced via an indirect word, then the descriptor is set up in a special page of its own. Depending upon the modification used in the indirect word, the descriptors may be in different segments.

Page Faults

You have control over a maximum of 13 page faults during the testing of any EIS instruction. These 13 pages have consistent meaning to eis_tester, even though for different tests they may actually be physically different pages in different segments.

The 13 pages are

1. Page 1 of the instruction area
2. Page 2 of the instruction area
3. Page containing indirect descriptor 1
4. Page 1 of data area 1
5. Page 2 of data area 1
6. Page 3 of data area 1
7. Page containing indirect descriptor 2
8. Page 1 of data area 2
9. Page 2 of data area 2
10. Page 3 of data area 2
11. Page containing indirect descriptor 3
12. Page 1 of data area 3
13. Page 2 of data area 3

Register Assignment

You can control the type of modification used by each EIS instruction tested. However, for each type of modification (depending upon the descriptor number) eis_tester assigns the register to be used. The specific use of pointers and registers is not under your control when using the eis_tester script input method.

Pointer registers not used during the instruction are set to null (77777|1). Index registers and the A and Q registers that are not used are set to 8191 decimal (17777 octal).

AR modification involves the use of a pointer register. Both descriptors and indirect words can use AR modification. A general rule followed by eis_tester is that AR modification implies the data referenced is in an external segment. The pointer registers used by eis_tester for the EIS instruction are

AR modification in a descriptor

descriptor 1	pr1
descriptor 2	pr2
descriptor 3	pr3

AR modification in an indirect word

descriptor 1	pr4
descriptor 2	pr5
descriptor 3	pr7

pr6 is used to point to your current stack frame and must be preserved in a valid state in order for any conditions to be signaled correctly.

Index register modification can be specified for descriptors and for indirect words. The effective offsets used for index modification are always set up by eis_tester in terms of words. For some descriptors, the value in the index register must be in units of characters. The character size also varies with the value of the ta field of the descriptor. The index registers assigned by eis_tester and the effective word offset they contain are given below.

Index register modification of a descriptor

descriptor 1	X1	1 word
descriptor 2	X2	2 words
descriptor 3	X3	3 words

Index register modification of an indirect word

descriptor 1	X4	4 words
descriptor 2	X5	5 words
descriptor 3	X7	7 words

RL modification can be specified for the descriptors of certain instructions. The value put in the register is specified by you. The register assigned is controlled by eis_tester. The following registers are used:

descriptor 1	A
descriptor 2	Q
descriptor 3	X6

Segments Used by eis_tester to Execute an Instruction

The execution of an instruction by eis_tester can involve up to seven segments:

etx	eti1	etd1
	eti2	etd2
	eti3	etd3

The notation etiX means eti1, eti2, or eti3, depending on which of the up-to-three descriptors or operands is of current interest. Similarly, etdX means etd1, etd2, or etd3.

The first list below states the possible segments in which various items can be located, while the second list states what segment a descriptor or an operand is in under various conditions.

ITEM	SEGMENTS		
	etx	etiX	etdX
Instruction word	x		
Indirect word pointing to descriptor	x		
Descriptor	x	x	
Operand	x	x	x

Is AR Used to Access Descriptor?	Is AR Used to Access Operand?	Descriptor Location	Operand Location
No	No	etx	etx
No	Yes	etx	etdX
Yes	No	etiX	etiX
Yes	Yes	etiX	etdX

eis_tester Printout

The eis_tester program prints a message noting the beginning of each instruction test. It also prints the number of this test. If there were errors, it prints the incorrect data or incorrect indicators.

If you do not specify -bf (see "Usage" above) then the data that eis_tester has set up for this instruction is printed before the instruction is executed. The following notes describe this printout:

1. Pointers enclosed in parentheses point to where the data is set up in the eis_tester segments.

2. If none of the pointer registers are used by the instruction, then none are printed. The same is true of the registers.

3. The names of the pages that take faults cannot be the names of all the pages specified in a page statement. See the last two complete examples at the end of this description for clarification.

4. If the first word of a data string does not begin at character 0 of a word, or if the string does not use all four characters of the last word, then the unused characters of the first and last words of the string are printed as blank characters.

5. The test string is not printed from one of the areas used by the instruction but rather from one of the buffers used by eis_tester.

6. The test and result strings are both padded by eis_tester with special characters. These special characters are not printed out in octal like the rest of the string; instead, each of these special characters is printed as three x's (xxx).

How to Call eis_tester

The eis_tester program is the main procedure in the EIS instruction tester. It calls et_test to parse the statements in your data file. It translates these statements into the data needed to build and test an EIS instruction in the external segment etx. After building the instruction, this procedure calls etx in order to execute the EIS instruction. When etx returns, the results of the EIS instruction are examined. The eis_tester program continues to build and test EIS instructions until there is no data left in the input file. The failure of one instruction only causes the termination of that one instruction test. Any remaining instructions specified in the input file are processed and tested.

How to Write Script Input Tests

The script input test consists of a series of eis_tester statements. The first statement in any test must be an inst statement. This statement signifies the beginning of one test.

An input script segment can contain several tests. All statements from the beginning of the inst statement to the beginning of the next inst statement (or, if none is found, to the end of the segment) are considered part of the same test.

The format of a statement is as follows:

```
name required_field {-control_args};
```

where:

1. name
   is the four-character statement name. There are four types of eis_tester statements:

inst	defines the instruction word and many control variables.
desc	defines a descriptor.
data	defines the data associated with a descriptor.
page	defines the page faults taken by the instruction.

   These statements are discussed in detail later in this document.

2. required_field
   is required information used by all but the page statement.

3. control_args
   are optional control arguments, explained in the individual statement descriptions.

Syntax and Metalanguage

All statements must end with a semicolon (;). There can be any number of blanks, tabs, and newline characters between any fields in the statement, including before the name field. Wherever blanks are permitted, there can also be comment fields. A comment field begins with a /* character pair and ends with the next */ character pair.

In this description, lowercase letters are used to indicate characters that are to be typed in for input to eis_tester. Uppercase letters are to be replaced with the desired character before the script is typed.

inst Statement

The inst statement defines the beginning of an eis_tester test. It is used to define all of the fields in the instruction word of the EIS instruction. It is also used to set up the following special control arguments:

1. Instruct eis_tester to execute this instruction several times.
2. Position the instruction within the instruction area.
3. Define an identifying string that is printed with the test.

An inst statement has the following format:

```
inst opcode_mnemonic {-control_args};
```

where:

1. inst
   is the four-character statement name.

2. opcode_mnemonic
   is the mnemonic name of a storage type EIS instruction.

3. control_args
   are optional and can be chosen from the following:

   -tbA
   turns on the truncation bit. The A is either y or n to signify whether or not the instruction is to take a truncation fault (y = yes, n = no). The default is n.

   -fb
   turns on the fill bit. The default is off.

   -pb
   turns on the plus sign bit. The default is off.

   -rb
   turns on the rounding bit. The default is off.

   -fcA
   defines the fill character to be the character specified by A. (No space between c and A and no quotes are permitted.)

   -mcA
   defines the mask character to be the character specified by A. (No space between c and A and no quotes are permitted.)

   -ln N
   defines the loop number as X. This is the number of times this instruction test is performed. The default is 1. The maximum value of X is 4.

   -io N
   defines the instruction offset. It is used to position the instruction relative to a page boundary. The default is 0. This places the instruction at word 0 of the second page of the instruction area. X indicates the number of words of the instruction to be placed in the first page of the instruction area. The maximum value of X is 3.

-nt "A...A"
defines a note. It can be used to identify each test. The term consists of a character string between quotes. Up to 32 characters can be used. No embedded quotes are allowed.

-bo AAA
defines a Boolean operator. AAA is the name of the operator. The names eis_tester has assigned to the Boolean operators are given below. Next to these names are the actual BOLR codes they represent.

```
zer  0000
and  0001
axr  0010
mov  0011
xra  0100
ra2  0101
xor  0110
orB  0111    Type in orB, where B is a space
nor  1000
nox  1001
iv2  1010
xrx  1011
inv  1100
xxr  1101
nan  1110
set  1111
```

-ir {terms}
is a multifield control argument that defines the correct state of the indicator registers after the EIS instruction has been executed. An -ir control argument can be followed by any number of specific terms. These terms can be in any order and can be separated by any number of skip fields. Each term is a two-character identifier of an indicator register bit.

A control argument of "-ir zr" means that the zero indicator is expected to be on at the end of the EIS instruction. Valid indicator register term values are

zr zero
ng negative
cr carry
ov overflow
eo exponent overflow
eu exponent underflow
om overflow mask
tr tally runout
pe parity error
pm parity mask
bm BAR mode (always turned on by eis_tester)
tn truncation
mw multiword instruction interrupt fault
ab absolute mode

If the script turns on eo, eu, or ov, then eis_tester automatically turns on the overflow mask bit in the expected indicator's result.

-mfX {terms}
is a multifield control argument that defines one mf field of the instruction. Some instructions do not have mf fields in the instruction word for all of their descriptors. The -mfX control argument is then used to specify any ar or reg modification in the descriptor itself. An example is the mvt instruction. X denotes which mf field is being defined. It must be from 1 to 3 and is associated with descriptor X. This descriptor number can be followed by up to four terms. All four terms are optional and can be specified in any order. The valid terms are

ar
rl L
idA
reg

The ar term

The ar term specifies that, for this descriptor, the address register modification is be used to access the operand. In Multics, it is called pointer register modification. The pointer assigned is prX. When this term is specified, the data referenced by this descriptor is placed in the segment etdX.

The rl L term

The rl L term specifies that, for this descriptor, the register length modification is be used. This term must be followed by a decimal number L, which specifies the character length of the data. The character size is defined within a desc statement (for 4-, 6-, or 9-bit characters) or inferred from the instruction mnemonic (for bit strings). This value is placed in the selected register, and the N field of descriptor X contains the register modification tag code. The registers assigned are

```
X = 1 - A
X = 2 - Q
X = 3 - x6
```

The idA term

The idA term specifies that descriptor X is to be referenced via an indirect word in the instruction. In the idA term, the A denotes what modification is to be used in the indirect word: a for address register, r for register, or b for both.

If no A character is given in the idA term, then there is no A modification in the indirect word.

ida

specifies that address register modification is to be used to access the descriptor. When this is specified, the descriptor is placed in the segment etiX.

The pointer registers assigned to the indirect word are

```
indirect word 1 => pr4
indirect word 2 => pr5
indirect word 3 => pr7
```

idr

specifies that register modification is to be used to access the descriptor. The indirect word is modified by index register 4, 5, or 7.

NOTE: This modification is in terms of words.

idb

specifies both a and r modification as described above.

The reg term

The reg term specifies that descriptor X is to be modified by an index register. The value in the index register is a character offset and is (X*4). The index register assigned is index register X. The value placed in index register X is dependent upon the type of instruction and the appropriate character size. It is in the following units:

WORDS
: for those descriptors that have no mf field in the instruction word

BITS
: for all bit string instructions

CHARS
: for all others. The actual units depend upon the character size. The default is a 9-bit character size.

If it is necessary to write a script in which the placement of the instruction, indirect words, descriptors, and operands in specific segments is important, the following list is help.

Script Elements Used in-mfX Fields	Descriptor Location	Operand Location
ar	etx	etdX
id ar	etx	etdX
ida ar	etiX	etdX
idr ar	etx	etdX
idb ar	etiX	etdX
id	etx	etx
ida	etiX	etiX
idr	etx	etx
idb	etiX	etiX

Examples of inst Statements

```
/*   Example 1.   */

inst      mlr       -nt  "Example 1 "

-fc*   /* Comments can go anywhere except inside a term */

-fb

-mf2  /* Note order is not important. */
         rl  3   id  ar  reg

-mf1  ar idb  reg  rl 3 ;     /* Statement must end with  ";"  */


/* Example 2. */    inst    cmpc        /* mnemonic name must
                                        *  be first term. */
-mf1  ar
-nt  " example 2"  -mf2  rl 3
-fc       /* Use escape to enter octal character   */
-ir  cr  zr ;       /*  Indicator  bm  is on by default.  */


/* Example 3.  */

inst      scm       -nt "scm examp."
-mc9
-ln  3              /* Make this test  3 times.  */
-io  2              /* Put instruction word and first descriptor
                    *  in page 1 of instruction area.  */
-mf1  reg ar
-mf2  ida;


/*        Example  4.  */

inst      ad3d
-mf3 ar  -mf2 reg -mf1 idr   /* -mfx items can be in any order */
-rb   -pb;


/*        Example 5. */

  inst csr -fb -bo and  -mf 2 rl  36;
```

desc Statement

The desc statements are used to specify certain fields in the descriptors. Each desc statement deals with only one descriptor. The fields in a descriptor not specifically set up by a desc or an inst statement are set to zero. If zero bits in all of the fields are needed, then no desc statement need be specified for that descriptor.

The -cp, -bp, and -cn fields of a desc statement interact with the -do field of the associated data statement. See the complete examples at the end of the eis_tester description for illustrations of the interactions.

In general, the order of the desc statements is not important, and the can be mixed in with any other statements. However, if the instruction is CMPC, SCD, SCDR, SCM, or SCMR, then the desc 2 statement cannot specify a -ta field. Descriptor 2 must use the value specified in descriptor 1. To use this feature, the desc 1 statement must precede the desc 2 statement.

A desc statement has the following format:

```
desc num {-control_args};
```

where:

1. desc
   is the four-character statement name.

2. num
   is the number of the descriptor. It must be 1, 2, or 3.

3. control_args
   can be chosen from the following:

   -cp N
   is used in bit string instructions to specify a (9-bit) character offset when developing an operand address where N must be a number from 0 to 3.

   -bp N
   is used in bit string instructions to specify a bit offset within a 9-bit character when developing an operand address where N must be a number from 0 to 8.

   -cn N
   is used in character string instructions to specify a character offset when developing an operand address where N must be a number from 0 to 7. The quantity of bits associated with each character (4, 6, or 9 bits) is specified by the N argument supplied with the -ta or -tn control argument.

eis_tester

-ta N
defines the alphanumeric character type where N must be 9, 6, or 4. The default value is 9.

-tn N
defines the type of numeric character where N must be either 9 or 4. The default value is 9.

-sd STR
is the sign and decimal type. The STR argument must be one of the following characters:

f - Floating point, leading sign
l - Leading sign, scaled
t - Trailing sign, scaled
n - No sign, scaled

-sf N
is the scaling factor where N is a signed (or unsigned) decimal number.

-ns N
is the number of characters or bits in a string where N is an unsigned decimal number. There is no default value.

-nn N
is the number of characters in a numeric string where N is an unsigned decimal number that must not be greater than 64. There is no default value.

EXAMPLES

```
/*        Example 1. */

desc  1   -ns 8  -ta 6  -cn 5;


/*        Example  2.  */

desc  3  -cp  2
-bp  /* Comments can come between control argument names
     *  and the term. */  5;
/* No -ns control argument.  This is valid if -mf3
*  control argument in inst statement
*  specified  rl  term.  */


/*        Example  3.  */

desc      2
-tn 4  -cn 3  -sd  n  /* No sign. */
-sf -100  -nn 12;
```

data Statement

The data statements are used to describe the data that a descriptor references. Every test requires at least as many data statements as there are descriptors for the EIS instruction being tested.

The eis_tester program can determine which descriptor references the result data. The data entered for this descriptor is not set up in the data area referenced by the descriptor. Instead, this data area is initialized to all zero bits. The input data is saved and used to test the result of the instruction. Some special notes about data statements are given below:

1. For those instructions that both read and write data into the same string (e.g., ad2d, sb2d), you must enter a data 3 statement that describes the resulting data referenced by descriptor 2. The data input via the data 2 statement is the data initially referenced by descriptor 2.

2. The data pointer for each descriptor is set by default to character 0 of word 0 of page 2 of the data area for that descriptor. You can adjust this data pointer by certain (9-bit) character offsets.

3. The input string defined by you is placed in the data area starting at the first character referenced by the effective data pointer. It is important to remember this. If the descriptor associated with this data area specifies that the first character of the string is not character 0 of the first word, then the missing data must be reserved when the input string is specified.

4. The -do field of a data statement interacts with the -cp, -bp, and -cn fields of the associated desc statement. See the complete examples at the end of the eis_tester description for illustrations of the interactions.

A data statement has the following format:

data num {-control_arg} data_fields;

where:

1. data
   is the four-character statement name.

2. num
   is the number of the data field. It must be either 1, 2, or 3. In some cases, a data 3 statement is valid even when there is no third descriptor. In this case, it is used to input test data. See the last complete example (csl instruction) at the end of this description. If the descriptor that points to this data does not use address register or register modification, then only offsets that are a multiple of 4 are accepted. The data used by EIS instructions is always string type data, and thus the input modes are limited to the two described below.

3. control_arg
   can be

   -do X
   where X must be a decimal integer from -128 to +4096 that represents a 9-bit character offset from character 0 of the middle page of the data area.

4. data_fields
   are the following types. They can be intermixed. The maximum size of the data is 1088 words (4352 characters).

   ASCII
   is an ASCII string. It must be enclosed in quotes. The maximum size of any one field is 256 characters. Quote characters can be entered in the string by expressing them as double quotes ("").

   OCTAL
   is a string of octal digits. The first nonoctal-digit-type character found indicates the end of a string of octal data. The converted octal string is padded on the right with zero bits to make it an integral number of 9-bit characters. For example, data 123 45 6 7777; becomes 123 450 600 777700.

   The repetition factor (XX), an unsigned decimal number enclosed in parentheses, can be used to specify the repetition of a field. Only the data field immediately following the repetition field is repeated.

EXAMPLES

```
/*        Example 1.  four characters of data starting at the
*         beginning of the default data area.  */

data 3    "abcd";


/*        Example 2.   Moves the same data field back two
*         characters.  This splits the string across a page.

*         NOTE:  The input string is the same even though it is
          entered differently.  */

data  2   -do -2    "ab"  "cd";


/*        Example 3.  The same as example 2 only it specifies
*         some of the data in octal.  */

data 1    "ab"  143144        -do -2;


/*        Example 4.  A string of:
*         "121212121212121212", that is 10 "12" strings.  */

data 2  "12" 061062 "1" "2"
061 062
(3) "12"
(3) 061062;


/*        Example 5.  The effective data address to be
*         word 1 of page 2 of the data area.  However, the cn
*         field of the descriptor specifies that the first
*         character of the word that is used is character
*         3.  Put some fill characters in the first
*         three characters.
*/

data 2    -do 4 "***"         /* Fill characters.  Not
                              *  referenced by the instruction. */
"abcd"    ;          /* The actual data string with which
                     *  the instruction works.  */
```

page Statement

The page statement is used to control page faults during the execution of the EIS instruction. The default case is that no page faults occur. The eis_tester program requires that you specify those pages on which faults are to be taken.

If you specify a page that is not actually used by the instruction (for example, the third page of a data area that has a one-character string), there is no harm. There is also no page fault.

All the pages used by an EIS instruction have been assigned names. For pages other than the two instruction area pages, the names can reference physically different pages. Their use by the EIS instruction is always the same.

The format of a page statement is

```
page {-control_args};
```

where:

1. page
   is the four-character statement name.

2. control_args
   specify what pages are to have page faults and be chosen from the following:

   -in1 -in2
   the two pages of the EIS instruction itself take a page fault.

   -id1 -id2 -id3
   the pages used by descriptors referenced via indirect words take a page fault.

   -d11 -d12 -d13
   the three pages of data referenced by descriptor 1 take a page fault.

   -d21 -d22 -d23
   the three pages of data referenced by descriptor 2 take a page fault.

   -d31 -d32
   the two pages of data referenced by descriptor 3 take a page fault.

   -all
   specifies that all of the pages defined for this instruction take a page fault. If other control arguments are entered along with the -all control argument, then the pages specified do not have page faults.

Running eis_tester with Other Users

If eis_tester is to be run while other users are on the system, it is not possible to positively guarantee that selected pages will not take a page fault. The "page -all;" statement causes eis_tester to flush all the pages of the etx, eti1, eti2, eti3, etd1, etd2, and etd3 segments out of main memory. Using "page -all -in2;" results in flushing all pages, touching page in2, and transferring control to etx. The touching of page in2 brings it into main memory. However, the overall system activity may be such that eis_tester loses control before reaching page in2, eis_tester and etx being displaced by pages for other users, control being returned to etx, execution continuing, and page in2 being no longer in memory. Then, when page in2 is needed, a page fault occurs. Therefore, a general guideline is: if eis_tester is run when other users are on the system, use the "page -all;" statement.

However, if eis_tester is to be run as the only user (nondaemon) process on the system and the "page" statement is not used, the pages should be in main memory when wanted. Some hardware problems may require running tests with and without page faults to isolate the problem. You should be aware that just because eis_tester attempts to avoid a page fault and the eis_tester output does not state that a page fault will occur does not necessarily mean that a page fault will not occur.

EXAMPLES

```
/*        Example 1. */

page  -in2  -id3 -d32
-d12  -d12  -d11  -id1;


/*        Example 2. */

page  -all;


/*        Example 3.   Take faults on  ALL pages  EXCEPT
*         pages in2  and  id3  */

page       -in2       -all       -id3;  /*  Notice order is not
                                         *   important.  */
```

EXAMPLES OF ACTUAL TEST SCRIPTS AND THEIR OUTPUT

```
/*                  mlr10
*
*         This test is the same as the test mlr3 except that
*         the descriptors use AR, REG, and RL modification
*         and use indirect descriptors.  The indirect
*         words use both REG and AR modification.
*/

inst      mlr       -nt  "10."                    -io 1
-mf1      rl  20
          ar        /* This puts the data in  etd1. */
          reg       /* Use index register 1.. */
          idb       /* This adds indirect descriptors.  Descriptors
                    *  go in segments  eti1 and eti2.  */

-mf2      idb
          rl  20
          reg
          ar;

desc 1    -cn 2;

desc 2    -cn 2;

data 1    -do  -20  "  "  (5) "abcd" ;

data 2    -do   -20  000 000  (5) "abcd" ; /* Fill for -cn 2
                                            *  must be zeros. */

page -in1  -in2
-d22 -d21
-d11 -d12
-id1 -id2
-id3 -d32;


et  mlr10  -nox

            /*The absence of any output from the
            *et mlr10 -nox input line means that the
            *script passes the validity checks that eis_tester performs.  */
```

```
et

TEST 1 (mlr)

EIS instruction:     ( 262|3777 )    Ind  Desc.
     000172100571
    - - - - -- - - -
     400034000114          ->       100007200005    ( 327|100 )
     500043000115          ->       200016200006    ( 330|100 )

Pointer Registers:  ( 262|20 )
     pr0 - pr3    777777|1  332|1763  333|1753  777777|1
     pr4 - pr7    327|40  330|30  777777|1  777777|1

Index Registers:     ( 262|70  )
      X0 - X7      17777     4    10    777   4   5 17777 17777
         A  000000000024       Q  000000000024

Test Indicators:     ( 262|111 )
     000000000200

This test takes  8  page faults.
     in1 in2 id1 d11 d12 id2 d21 d22

data field 1         ( 332|1773 )
     000000141142  143144141142  143144141142  143144141142
     143144141142  143144

data field 2         ( 333|1773 )
     Result data field initialized to all zero bits.

test  data           ( 262|15776 )
     xxxxxxxxxxxx  xxxxxxxxxxxx  040040141142  143144141142
     143144141142  143144141142  143144141142  143144xxxxxx
     xxxxxxxxxxxx  xxxxxx

/*        Test  mvt  instruction.  */

inst      mvt       -nt "3"   -fc       /* Char is octal 1. */
-mf1  rl 3  ar  reg  idr
-mf2  ar reg ida
-mf3  reg  ar;

desc 2  -ns 8;

data 1    -do -2    003 002  001;
```

```
data 2    -do -6    "321111"  "11";

data 3    -do  -1   "0"  "123";


page  -all  -in2;


et

TEST 1 (mvt)

EIS instruction:     ( 262|4000 )   Ind  Desc.
    - - - - -- - - -
     001132160571
     051774000014          ->       100007000005    ( 262|52000 )
     500050000100          ->       200016000010    ( 330|100 )
     300025000113

Pointer Registers:   ( 262|20 )
     pr0 - pr3    777777|1  332|1767(18)  333|1756(18)  334|1747(27)
     pr4 - pr7    777777|1  330|30  777777|1  777777|1

Index Registers:     ( 262|70  )
     X0 -X7         17777     4   10    777   4   5 17777 17777
         A  000000000003      Q  000000017777

.Test Indicators:    ( 262|111 )
     000000000200

This test takes  11  page faults.
     id1 d11 d12 d13 id2 d21 d22 d23 d31 d32 d33

data field 1         ( 332|1777(18) )
           003002  001

data field 2         ( 333|1776(18) )
     Result data field initialized to all zero bits.

data field 3         ( 334|1777(27) )
              060  061062063

test  data           ( 262|15776 )
     xxxxxxxxxxxx  xxxxxxxxxxxx  063062061061  061061061061
     xxxxxxxxxxxx  xxxxxxxxxxxx
```

NOTES

A standard set of scripts is provided that can be used with eis_tester. If CPU problems with the EIS instructions are suspected, these scripts should be run. The ets segment's standard location in the storage hierarchy is >tools>ets, but an installation can locate ets somewhere else.

EXAMPLES

```
/*  An example to illustrate the interaction between
 *  the "-do" and the "-cn" fields.
 */
inst mlr   -ir tn  -nt "-do and -cn interaction"
     -mf1 idb ar reg    -mf2 idb ar reg;
/*  The uppercase letters in the two data statements could have been
 *  typed in as  "ABCDEFGH"  and  "CDEF"  but were typed in as they are
 *  shown so that explanatory remarks could be placed on the adjacent
 *  lines.  The symbols used above and below the desc and data lines
 *  mean:
 *        P       The boundary of a page, and hence, also a word
 *                boundary.
 *        S----S  The operand string portion of the data field.
 *        W       A word boundary.
 *                              W                P                W   */
desc 1 -cn 2 -ns 6;data 1 -do -5 "A" "B" "C" "D" "E" "F" "G" "H" "I" ;
/*                                    S------------------S
 *  Each uppercase letter in the above data statement occupies 9 bits.
 *  Note that the data field for the first operand starts five 9-bit
 *  bytes to the left of a page boundary.  This is due to the "-do -5"
 *  field.  However, the operand string excludes the first two bytes
 *  of the data field, because of the "-cn 2" field.
 *
 *                                       P             W           */
desc 2 -cn 3 -ns 4;data 2 -do -1     000 000 000 "C" "D" "E" "F";
/*                                             S-----------S
 *
 *  The data field for the second operand starts one 9-bit byte to the
 *  left of the page boundary due to the "-do -1" field.  The "-cn 3"
 *  field results in the operand skipping over the first three bytes of
 *  the data field.  Another way to specify the CDEF string to fall
 *  where it does would be to use these desc 2 and data 2 statements:
 *      desc 2 -ns 4;  data 2 -do 2 "CDEF";
 *  The "-cn 3" and  000 000 000  were used to show how to do it when
 *  the person writing the script wants to use the CN field in the
 *  second descriptor.     */
page -all;
```

```
COMMENT:  The output from running eis_tester with the above script is shown
COMMENT:  below.  Explanatory remarks have been inserted in the
COMMENT:  output, on the lines that start with COMMENT:

ET
TEST   1 (mlr)

Test Description:  -do and -cn interaction

Eis instruction:     ( 340|4000 )   Ind  Desc.
    - - - -- - - -
     000132100531
     400034000114          ->       100007400006    ( 341|100 )
     500043000115          ->       200016600004    ( 342|100 )

Pointer Registers:  ( 340|20 )
     pr0 - pr3    77777|1  344|1766(27)  345|1757(27)  77777|1

COMMENT:  The value in the parentheses following a word offset, which
COMMENT:  is in octal, is the bit offset, in decimal.

     pr4 - pr7    341|40  342|30  77777|1  77777|1

Index Registers:     ( 340|70  )
      X0 - X7   17777   4   10   17777   4   5   17777   17777
         A  000000017777      Q  000000017777

Test Indicators:     ( 340|111 )
     000000000300

This test takes 7 page faults.
     in2 id1 d11 d12 id2 d21 d22

COMMENT:  Page d21 is included here because data_field_2 crosses the
COMMENT:  boundary between the first and second pages of the etd2
COMMENT:  segment.  However, because of the "-cn 3" field, operand_2
COMMENT:  actually resides in only the second page.  Therefore, the
COMMENT:  first page is not be touched, and no page fault occurs
COMMENT:  for page d21.
```

```
data field 1         ( 344|1776(27) )
             101  102103104105  106107110111
COMMENT:               S-----------------S
COMMENT:  It is true that data_field_1 begins in bit 27 of word 1776.
COMMENT:  However, because of the "-cn 2" field, operand_1 begins
COMMENT:  with bit 9 of word 1777.  The address development
COMMENT:  (in octal) for the start of operand_1 is:
COMMENT:
COMMENT:  ITEM   SEGMENT|WORD   9-BITBYTE
COMMENT:  ----   ------------    --------
COMMENT:  desc_1                7    2
COMMENT:  pr1            344|1766    3
COMMENT:  x1            04
COMMENT:       =         344|1775   11
COMMENT:  which is       344|1777    1    or    344|1777(9)
COMMENT:  If the same calculations are carried out for the second
COMMENT:  operand, it is seen that the data field starts in
COMMENT:  one page but the operand starts in the next page.
COMMENT:  Refer to the script line above that contains the
COMMENT:  "desc 2" and "data 2" statements and then examine the
COMMENT:  adjacent lines.

data field 2         ( 345|1777(27) )
     Result data field initialized to all zero bits.

test  data           ( 340|15776 )
     xxxxxxxxxxxx  xxxxxxxxxxxx  000000000103  104105106xxx
COMMENT:                                      S------------S
COMMENT:  The xxxxxx represent fill supplied by eis_tester.  The nine
COMMENT:  leading octal zero digits are present because they were
COMMENT:  supplied in the "data 2" statement.
     xxxxxxxxxxxx  xxxxxxxxx


/*  An example to illustrate the interaction between
 *  the "-do" and "-cp" and "-bp" fields.     */
inst csl  -bo or  -nt "-do and -cp and -bp interaction"
     -mf1 idb ar reg    -mf2 idb ar reg;
desc 1 -cp 2 -bp 3 -ns 30;
/*  The symbols used above and below the data lines mean:
 *        P       The boundary of a page, and hence, also a word
 *                boundary.
 *        S----S  The operand string portion of the data field.
 *        W       A word boundary.
 *                      W               P           */
```

```
data 1 -do -7 123 456 701 020 203 040 123 456 765;
/*                    S----------S
 *  Data_field_1 starts seven 9-bit bytes before the page boundary,
 *  due to the "-do -7" field.  The "-cp 2" field causes the processor
 *  to skip over the first two bytes (123 456 octal), so that
 *  operand_1 starts somewhere in the 701 octal byte.  The
 *  "-bp 3" field causes the processor to skip over the first three
 *  bits (7 octal) of the 701 byte, thereby starting at bit 30
 *  (bits numbered 0-35) of the next to last word of a page.
 */
desc 2 -cp 1 -bp 6 -ns 30;
/*         Pxxx            W               W         */
data 2 -do 1    432 103 030 405 050 765 432 101;
/*                   S----------S
 *  The "Pxxx" above the "data 2" statement is intended to
 *  indicate that the page boundary is three octal digits (the xxx)
 *  i.e., nine bits, before the start of data_field_2, as
 *  specified by the "-do 1" field.  The "-cp 1" field specifies
 *  skipping over the first 9-bit byte, to the 103 octal byte.
 *  The "-bp 6" field specifies skipping the first six bits of
 *  that byte, to the octal 3, which begins in bit position 24
 *  (of 0-35) in the first word of a page.
 *                        W               W         */
data 3          432 103 132 425 354 765 432 101;
/*                   S----------S
 *  The "data 3" statement is used because the  csl  instruction
 *  stores its result in the same bit locations from which the
 *  second operand was fetched.  No "-do", "-cp", or "-bp" fields
 *  are needed for the "data 3" statement because eis_tester
 *  associates the attributes of data_field_2 and operand_2 with
 *  the data supplied by the "data 3" statement.
 */
page -all;

ET
TEST   1 (csl)

Test Description:  -do and -cp and -bp interaction

Eis instruction:     ( 334|4000 )    Ind  Desc.
    - - - -- - - -
     007132060531
     400034000114          ->       100007430036    ( 335|100 )
     500043000115          ->       200016260036    ( 336|100 )

Pointer Registers:  ( 334|20 )
     pr0 - pr3    77777|1  340|1766(9)  341|1760(9)  77777|1
     pr4 - pr7    335|40  336|30  77777|1  77777|1
```

```
Index Registers:     ( 334|70  )
      X0 - X7   17777   44   110   17777   4   5   17777   17777
         A  000000017777      Q  000000017777

Test Indicators:     ( 334|111 )
     000000000200

This test takes 7 page faults.
     in2 id1 d11 d12 id2 d22 d32
COMMENT:  Page d12 is listed here because data_field_1 is in both
COMMENT:  pages 1 and 2.  However, operand_1 is in only page 1,
COMMENT:  so a page fault does not occur for page d12.

data field 1         ( 340|1776(9) )
        123456701  020203040123  456765
COMMENT:       S----------S

data field 2         ( 341|2000(9) )
        432103030  405050765432  101
COMMENT:     S----------S
COMMENT:  The address development for the start of operand_2 is
COMMENT:  shown below.  For x2, 110 octal = 72 decimal = 2 words
COMMENT:  and no bits.
COMMENT:                                    9-BIT BYTE          BIT IN
COMMENT:              SEGMENT|WORD          IN WORD             BYTE
COMMENT:  ITEM          (OCTAL)             (BINARY)            (BINARY)
COMMENT:
COMMENT:
COMMENT:  desc_2              16            01                  0110
COMMENT:  pr2           341|1760            01                  0000
COMMENT:  x2                  2             00                  0000
COMMENT:                ------------------------------------------
COMMENT:       =        341|2000            10                  0110
COMMENT:
COMMENT:  which is segment 341, word 0 of the second page, 9-bit
COMMENT:  byte number 2 (numbering is 0-3), and bit number 6
COMMENT:  (of 0-8), i.e.,  341|2000(24).

test  data           ( 334|23776 )
     xxxxxxxxxxxx  xxxxxxxxxxxx  432103132425  354765432101
COMMENT:                                S----------S
COMMENT:  The leading fill of 43210  and the trailing fill of
COMMENT:  765432101 were not affected by the execution of the
COMMENT:  instruction, proving that bits outside the operand
COMMENT:  strings did not enter into the instruction's execution.
     xxxxxxxxxxxx  xxxxxxxxxxxx
```

Name: exercise_disk

SYNTAX AS A COMMAND

```
exercise_disk disk_type volume_id {-control_args}
```

FUNCTION

exercises a disk drive. Maximal arm motion occurs all over the disk, and data is written and read back later for checking at each point. This activity can be used to make unstable drives fail repeatedly.

ARGUMENTS

disk_type
a valid Multics disk device type (e.g., d451, d500, and d501).

volume_id
the label of the disk pack on which the test is to be run.

CONTROL ARGUMENTS

-write_read, -wr
writes a known pattern over the entire disk pack, and then reads this information back for checking purposes. This is the default.

-write, -w
writes a known pattern over the entire disk pack. The default is -write_read.

-read, -r
reads back the information on the disk pack, for checking purposes. The default is -write_read.

-device STR, -dv STR
specifies the device on which you want to run the test, where STR can be dska_02, dskb_13, etc.). Either this control argument or the assign_resource (ar) command must be used to attach an I/O disk.

-no_data_compare, -ndc
makes no data compare on the read pass; only errors detected by the hardware are reported. This enables testing of a disk pack without knowing what data is recorded on it. The default is to compare the data with a known pattern.

-random
the test performs random rather than sequential seeks; the test takes several hours. This is the default.

-sequential, -sq
the test runs sequentially, writing and reading from sector 0.

-alternate_track, -altrk
removes the alternate track area of the disk from the test parameters. The default is to use the entire pack.

-from M, -fm M
sets the lower limit of the range of addresses to be tested to M, where M is a decimal integer specifying a valid cylinder number for the device to be tested.

-to N
sets the high limit of the range of addresses to be tested to N, where N is a decimal integer specifying a valid cylinder number for the device to be tested.

NOTES

The exercise_disk command requests the mounting of a scratch pack.

The assign_resource command must be used in conjunction with this command to exercise a given drive. Such drives must be configured as user I/O drives (nonstorage system) via the UDSK CONFIG card (see the description of the UDSK CONFIG card in the MOH) or by the use of the set_drive_usage (sdu) initializer command.

When the -from or -to control arguments are used, testing is confined to the range of addresses specified. The seek pattern used in this mode is from inner cylinder to outer cylinder, with M incrementing to N or the maximum address of the device, and N decrementing to M or cylinder zero. When M reaches its inner limit, the pattern is repeated. Testing continues until stopped by the user, by hitting the break key and then typing the release (rl) command.

Name: fnp_data_summary

SYNTAX AS A COMMAND

```
fnp_data_summary {fnp_names} {-control_args}
```

FUNCTION

reports error statistics recorded in the syserr log by the poll_fnp command. The statistics reported include parity errors for all channels and various counters for synchronous channels whose interpretation depends on the line type of the channel. Only nonzero statistics are reported.

ARGUMENTS

fnp_names
are the names of FNPs for which statistics are to be reported. If no fnp_names are specified, statistics are reported for all FNPs for which any nonzero statistics are available.

CONTROL ARGUMENTS

-expand
displays in expanded form every entry in the syserr log containing statistics for the specified FNP(s).

-extend, -ext
appends the output of the command to the end of the output_file if it already exists. This control argument can be specified only if -output_file (below) is specified.

-from DATE_TIME, -fm DATE_TIME
starts scanning the syserr log from the time specified by DATE_TIME, which must be a character string acceptable to convert_date_to_binary_. The default is to start at the beginning of the log.

-for TIME
restricts the scan of the syserr log to an interval of length TIME, where TIME is a character string representation of a time interval in a form acceptable to convert_date_to_binary_. This control argument is incompatible with -to (below). The default is to continue the scan up to the end of the log.

-output_file {PATH}, -of {PATH}
directs output to a file. If PATH is specified, it is the pathname of the output file; otherwise, output is sent to fnp_data_summary.output in the current working directory. If -output_file is not specified, the default is to direct output to your terminal.

-to DATE_TIME
ends the scan of the syserr log at the time specified by DATE_TIME, which must be a character string acceptable to convert_date_to_binary_. This control argument is incompatible with -for (above). The default is to continue the scan to the end of the log.

Name: io_error_summary

SYNTAX AS A COMMAND

```
io_error_summary {-control_args}
```

FUNCTION

scans the syserr log and summarizes I/O errors in a brief report.

CONTROL ARGUMENTS

-cylinders, -cyl
separates the disk device error by cylinder and head. Only disk_control can be separated.

-from DT, -fm DT
starts scanning the log at the date/time given.

-to DT
stops scanning the log at the date/time given.

-for T
computes the ending time from the starting time, where T is a relative time (such as "1 hour").

-device STRs, -dv STRs
reports information for the device(s) named, where STRs are device types ("prt") or device names ("prtb").

-detailed_status, -dtst
displays detailed status if available.

-hex_detailed_status, -hxdtst
displays detailed status in hexidecimal if available.

-tape_data_bit_in_error, -tdbie
displays the data bit(s) in the detailed status that were in error.

-status status_list, -st status_list
reports information for the IOM status listed, where status_list is the IOM major and substatus ("0310" or "4310").

NOTES

If -from DT is not specified, the scan starts with the earliest message in the syserr log. The ending time can be specified by using -for, or -to, but not both. If both are omitted, the scan terminates with the last message in the log. All dates and times must be in a format acceptable to convert_date_to_binary_ described in the Subroutines manual.

You must have re access to audit_gate_ and r access to the permanent_syserr_log and config_deck segments to use this command.

Name: list__proc__required

SYNTAX AS A COMMAND

```
list_proc_required {-control_args}
```

FUNCTION

determines the group of CPUs on which the invoking process can be run or the default group of CPUs for all processes that have not requested specific CPUs.

SYNTAX AS AN ACTIVE FUNCTION

```
[list_proc_required {control_args}]
```

CONTROL ARGUMENTS

-priv
indicates that this command applies to the default group of CPUs for processes that have not requested specific CPUs. If omitted, this command applies to the group of CPUs for the invoking process only.

NOTES

When invoked as a command without the -priv control argument, list_proc_required indicates that the set of CPUs needed for this process is the system default by printing "(default)" following the list of CPUs. This information is not provided when list_proc_required is invoked as an active function. If invoked as an active function, it returns a string of CPU tags that represent the group of CPUs requested (e.g., "ABCF").

This command prints the list of CPUs required as an uppercase string. If invoked as an active function, this returned list of CPU tags is in uppercase.

ACCESS REQUIRED

It requires access to phcs_ or metering_gate_.

EXAMPLES

This command is most useful when used in conjunction with the set_proc_required command to verify that the restriction specified in an earlier invocation of set_proc_required is still in operation. The effect of set_proc_required can be canceled by the system because of dynamic reconfiguration without notification to the process affected. If the following set of commands are input:

```
set_proc_required A

{other commands}

list_proc_required
```

an output of "A" from list_proc_required indicates that all commands between the set_proc_required and the list_proc_required were run entirely on CPU a. Any other output indicates that the effect of the set_proc_required has been canceled due to dynamic reconfiguration.

Name: load_mpc

SYNTAX AS A COMMAND

```
load_mpc {mpc_name} {-control_args}
```

FUNCTION

loads ITRs or application firmware or both into MPCs.

ARGUMENTS

mpc_name
: is the name of the MPC to be tested or reloaded or both. This name must appear on an MPC card in the config deck. If this argument is omitted, the -channel control argument must be given.

CONTROL ARGUMENTS

channel channel_name, chn channel_name
: specifies a channel name, where channel_name is of the form {iomtag} channel_no (for example, a14). The iomtag field must be the tag of a configured IOM and is required on multiple IOM systems. The channel_no field is an octal channel number. If this control argument is used, the mpc_name argument is optional. If both are used, the channel must be connected to the mpc specified.

-itr
: loads only the ITRs; the standard firmware is not reloaded.

-firm
: loads ony the standard firmware; ITRs are not run.

-revision RV, -rev RV
: specifies which revision firmware is to be loaded, where RV is a 2-character firmware revision code. If multiple revisions exist and this argument is omitted, you are queried as to which revision to load.

-time, -tm
prints timings for each program loaded into the MPC.

-brief, -bf
withholds printing of the names of the programs as they are run.

NOTES

By default this command suspends I/O on all devices connected to the selected MPC, resets the controller, runs all the known ITRs, reloads the standard firmware (including device routines for urmpc), and restores I/O on all devices connected to the controller.

If any abnormal conditions occur, the program displays the status that occurred, and stops. I/O is left in a suspended state, because the MPC has been left in an unusable state. In order to return the controller to operation, it is necessary to restore the firmware, using either this command or TOLTS (documented in the Online T&D manual).

This command can be used on disk MPCs only if they are fully cross barred.

Firmware and ITR modules are found in the Test and Diagnostics (T&D) deckfile created by the load_tandd_library command (also documented in the T&D manual).

Name: mc_trace, mct

SYNTAX AS A COMMAND

```
mc_trace path {-control_args}
```

FUNCTION

gives a snapshot of machine conditions and history registers (resulting from hardware faults and interrupts) incurred while executing another Multics command or subroutine.

ARGUMENTS

path
is the absolute or relative pathname of the segment that is to be traced.

CONTROL ARGUMENTS

-all
captures machine conditions and history registers for every fault and interrupt that occurs in your process. This control argument cannot be used with -hc or the path argument.

-brief, -bf
suppresses printing your prompt "-->".

-buffer N, -buf N
sets the machine condition trace buffer size to N, where N is a decimal integer value from 1 to 16, and represents the buffer size in units of 1024 words (1K). The default buffer size is 5K words.

-hc SEG
captures machine conditions and history registers for faults and interrupts that occur in the hardcore segment SEG while your process is in execution. SEG can be a hardcore segment name or number. This control argument cannot be used with -all or the path argument.

NOTES

This command initiates the segment specified by the path argument, and creates the machine condition trace buffer in your process directory. The number of machine conditions and history register sets that can be stored is directly related to the size of the trace buffer. There is an approximate 8 to 1 ratio of machine conditions to history registers (e.g., in a 5K buffer there would be storage for 79 sets of machine conditions and 10 sets of history registers allowing room for a trace buffer header). The trace buffer is temporarily "wired" (i.e., the segment remains in main storage and is not subject to removal by the dynamic paging mechanism). The hardcore snapshot or trace mechanism is then enabled and mc_trace goes into a request loop after printing "-->" as your prompt on the error_output switch. The valid user reponses while in this request loop are as follows:

1. .
prints out the command name "mc_trace" on the user_output switch.

2. .q
turns the hardcore snapshot mechanism off, unwires the machine condition buffer, and returns to Multics command level.

3. ..<command>
calls the Multics command processor and executes <command> as a Multics command (e.g., ..who).

4. .rpt n <command>
calls the Multics command processor to loop n times, executing the specified Multics command <command>; n is an integer from 1 to 99999999 (e.g., .rpt 10 who).

5. .pmc m n
displays machine conditions in octal starting with machine condition set m for n sets. The integer m therefore represents a negative index from the last set of machine conditions stored (e.g., the request ".pmc 8 2" would be interpreted to mean, "display two sets of machine conditions starting from the last machine conditions stored at position 8"). If n is not specified, then all machine conditions starting at m to the last machine conditions stored are displayed. If neither m nor n are specified, all sets of machine conditions are displayed.

6. .pmci m n
   same as .pmc above except that the machine conditions are displayed in interpreted format.

7. .pscu m n
   same as .pmc above except that only the System Control Unit (SCU) data for the specified number of machine conditions is printed, displayed in interpreted format.

8. .hr m n
   displays history registers in octal, starting with history register set m for n sets. The variables m and n are defined as in .pmc above.

9. .hrou m n
   same as .hr above except that only the Operations Unit (OU) history register is displayed in octal.

10. .hrcu m n
    same as .hr above except that only the Control Unit (CU) history registers are displayed in octal.

11. .hrdu m n
    same as .hr above except that only the Decimal Unit (DU) history registers are displayed in octal.

12. .hrau m n
    same as .hr above except that only the Appending Unit (AU) history registers are displayed in octal.

13. .hranl m n
    same as .hr above except that the specified number of history registers are displayed in interpreted format.

14. .hrlgd
    produces a list of abbreviations used with the .hranl request above.

The mc_trace command invokes a condition handler for the "any_other" condition. When any unusual system condition is encountered, a message indicating the condition that was raised is displayed on the error_output I/O switch, and control is passed to the request loop. At this time, any of the valid requests described above can be entered. For further information on system conditions, refer to the *Programmer's Reference Manual*, Order No. AG91).

To use mc_trace, you must have re access to phcs_.

EXAMPLES

Assume that you have written a program that generates an op_not_complete fault while executing a csl (combine bit strings left) EIS instruction in a particular sequence (e.g., descriptors fall on page boundaries). This is clearly a hardware problem, but because it only occurs when a particular set of events take place, it is very difficult for the field engineer to trouble-shoot. For simplicity, call this program onc_csl. To run this program under control of mc_trace, you execute the following sequence of commands:

```
mc_trace onc_csl
     start trace with default buffer size of 5K words.
--> ..set_proc_required a
     run with only one processor, in this case processor a.
--> ..onc_csl
     execute the program onc_csl
op_not_complete condition raised, enter command.
```

At this point, the op_not_complete condition has occurred, and the machine condition history for the last 103 machine conditions should be preserved in the machine condition buffer. You can now selectively display these machine conditions.

```
--> ..set_proc_required
      reset set_proc_required (run on any processor now)
--> ..file_output onc_trace
      direct the output from the user_output I/O switch to the file named
      onc_trace created in your working directory.
--> .pmc
      display the entire machine condition buffer. In this case, the output goes
      to the onc_trace file.
--> .pmci
      additionally display the machine conditions in interpretive format.
--> .pscu
      also display only SCU data in interpretive format.
--> .hr 1
      display history registers from last fault (in this case, the op_not_complete
      fault) in octal format.
--> .hranl 1
      display a composite analysis of the last set of history registers.
--> .hranl 2 1
      display a composite analysis of the next to last set of history registers.
--> ..revert_output
      direct the output from the user_output I/O switch from the onc_trace
      file back to your terminal.
--> ..dprint onc_trace
      print the onc_trace file on a remote printer.
--> .q
      return to Multics command level.
```

If the op_not_complete fault does not occur on a consistent basis and it is suspected that it only occurs randomly when a particular sequence of page faults and interrupts occurs, you can use another program called "flush", which generates heavy paging activity, and can loop on these commands several times by executing the following line in place of the onc_csl command line:

```
--> .rpt 99999999 flush;onc_csl
op_not_complete condition raised, enter command
-->
```

At this point, you can proceed as in the above example and print the machine conditions to a file or display them on the terminal.

OUTPUT PRODUCED WITH THE .PMC, .PMCI, AND .PSCU REQUESTS

.pmc REQUEST

The .pmc request produces an octal dump of the machine conditions, separated by the logical data in the machine conditions (e.g., pointer registers, processor registers). The format is dependent on the state of the user_output I/O switch. If the user_output I/O switch is attached to a file or a terminal with a line length greater than or equal to 104 characters, then the output is formatted in lines of eight octal words per line. If the user_output I/O switch is attached to a terminal with a line length less than 104 characters per line, then the output is formatted in lines of four octal words per line.

```
*****Machine Conditions at mc_trace_buffer|2410*****

Pointer Registers
000035000043 004646000000 000017000043 000000000000
000062000043 005362000000 000135000043 000000000000
000014000043 006712000000 000062000043 000000000000
000062000043 004060000000 000356400043 000000000000

Processor Registers
011127005716 000000000031 060614000030 000015000720
000000000012 000000000004 000000000000 001714442000

SCU Data
000113050202 000000000043 400356000004 000000540000
044571000200 000000000400 700000100440 000140100540

Software Data
121040000012 420040000006 000000000000 000000000000
000000000000 000000000000 000000104422 532243555724

EIS Pointers and Lengths
000400000000 000400000000 000000000070 004077777774
000102000030 000000000000 030356000004 000000000077
```

.pmci REQUEST

The .pmci request displays the machine conditions in an interpreted format, as shown below.

```
*****Machine Conditions at mc_trace_buffer|2410*****

pr0 (ap) 35|4646 bound_sss_wired_|4646
pr1 (ab) 17|0    sst_seg|0
pr2 (bp) 62|5362 pds|5362
pr3 (bb) 135|0   dirlockt_seg|0
pr4 (lp) 14|6712 as_linkage|6712
pr5 (lb) 62|0    pds|0
pr6 (sp) 62|4060 pds|4060
pr7 (sb) 356|0   complex_decimal_op_|0 (bound_pl1_runtime_|0)

x0  11127   x1   5716   x2      0   x3     31
x4  60614   x5     30   x6     15   x7    720
a 000000000012  q 000000000004  e 0
Timer reg - 1714442, Ring alarm reg - 0

SCU Data:

   240    000113050202 000000000043 400356000004 000000540000
          044571000200 000000000400 700000100440 000140100540

(DF1) Page Fault (43)
By: 113|44571 bound_file_system|44571
Referencing: 356|0 complex_decimal_op_|0  (bound_pl1_runtime_|0)
On: cpu a (#0)
Indicators:  ^bar
APU Status:  sd-on, pt-on, ptw
CU Status:   rfi
Instructions:
  246  700 000 100 440  mlr    (rl),(),fill(700)
  247  000 140 100 540  mlr    (pr,rl),(pr,rl),fill(000)

Time stored: 05/31/77  1355.3 mst Tue (1044225322435555724)
Ring:          0

EIS Pointers and Lengths:

   260    000400000000 000400000000 000000000070 004077777774
          000102000030 000000000000 030356000004 000000000077
```

.pscu REQUEST

The .pscu request displays only the SCU data from the machine conditions in an interpreted fashion, as shown below.

```
*****Machine Conditions at mc_trace_buf|2410*****

SCU data at mc_trace_buf|240

   240     000113050202 000000000043 400356000004 000000540000
           044571000200 000000000400 700000100440 000140100540

(DF1) Page Fault (43)
By: 113|44571 bound_file_system|44571
Referencing: 356|0 complex_decimal_op_|0  (bound_pl1_runtime_|0)
On: cpu a (#0)
Indicators:  ^bar
APU Status:  sd-on, pt-on, ptw
CU Status:   rfi
Instructions:
  246  700 000 100 440  mlr    (rl),(),fill(700)
  247  000 140 100 540  mlr    (pr,rl),(pr,rl),fill(000)
```

OUTPUT PRODUCED BY THE .BR REQUEST

The .hr request produces an octal dump of the history registers. The output is separated by the history register type being dumped. The format is dependent on the state of the user_output I/O switch. If the user_output I/O switch is attached to a file or a terminal with a line length greater than or equal to 104 characters, then the output is formatted in lines of eight octal words per line. If the user_output I/O switch is attached to a terminal with a line length less than 104 characters per line, then the output is formatted in lines of four octal words per line. If the .hrou, .hrcu, .hrdu, or .hrau requests are selected, only the requested history register type is dumped.

*****History Registers at mc_trace_buf|210*****

Operations Unit (OU) History Registers
315000315100 137777012705 450000450300 177777012707
255000255300 177777012710 221000221100 136777012711
220000221340 114777012717 220000220100 135777012720
740000740300 175777012721 741000741300 176777012722
450000450300 177777012725 446000446300 177757012726
720000720500 135777012727 440000440300 175777012730
450000450300 177777012731 621000621500 122777000051
431210431100 123777000052 431210431000 037777000054

Control Unit (CU) History Registers
700137352120 000046050200 300117352120 027342050020
200117352000 027220050024 200017252100 027574242100
700137352120 000050050200 300137352120 027344050021
200117352000 027224050024 200017252100 027576242100
600137621100 000052050200 200117621100 027566050014
200017431007 020000042014 700137370120 000054042200
300137370120 027524050021 200117370000 034304050024
300017352120 034314042020 300005352046 034314402002

Decimal Unit (DU) History Registers
737757037737 744243410017 737757037737 744243410017
777757037737 744243410017 737757037717 744243410017
737757037737 744243410017 777757037737 744243410017
737757037737 744243410017 737757037717 744243410017
777757037737 744243410017 737757037737 744243410017
737757037737 744243410017 777757037717 744243410017
737757037737 744243410017 777757037737 744243410017
737757037737 744243410017 777757037717 744243410017

Appending Unit (AU) History Registers
002342006066 027555724775 003576007262 024520464775
002342006066 027553424775 002346001420 000272204775
002342006066 027555744775 003576007262 024520504775
002342006066 027553444775 002346001420 000272244775
002342006066 027555764775 003576007262 024520524775
003576001420 000275664775 003576001420 000200004775
003576007262 024520544775 002342006066 027555244775
002556001420 000343044775 002552007370 026403144000

OUTPUT PRODUCED BY THE .HRANL REQUEST

The .hranl request produces a composite analysis of the history registers in the trace buffer. The output produced is dependent on the state of the user_output I/O switch. If the user_output I/O switch is attached to a terminal with a line length less than 104 characters, the output appears as below. If the user_output I/O switch is attached to a file or a terminal with a line length greater than or equal to 104 characters, then the octal representation of the history registers is displayed in addition to the sample below:

```
*****History Registers at mc_trace_buf|210*****

Composite Analysis of History Registers

 HR                          c
id##   IC_____ opcd__ tag_ y seg#_ offset__ mc flags
CU 1           epp2   n*   i   415       46  4 pi pa ri ic wi it
AU 2                          7  6   336046  4 ap sm pm
CU 2             "    n*   n   234    27342  4 pa ri ic it cl
AU 3                         12  5   533342  4 ap sm pm
CU 3             "        d   234    27220  4 pa ic cl dr
AU 4                                 27220  4
CU 4        46 spri2      o   234    27574 24 pa it cs
AU 5                         12  5   533574  4 ap sm pm
CU 5        47 epp2   n*   i   415       50  4 pi pa ri ic wi it
AU 6                          7  6   336050  4 ap sm pm
CU 6             "    n*   n   234    27344  4 pa ri ic wi it cl pb
AU 7                         12  5   533344  4 ap sm pm
CU 7             "        d   234    27224  4 pa ic cl dr
AU10                                 27224  4
CU10        50 spri2      o   234    27576 24 pa it cs
AU11                         12  5   533576  4 ap sm pm
CU11        51 eax1       i   415       52  4 pi pa ic wi it
AU12                          7  6   336052  4 ap sm pm
CU12             "        d   415    27566  4 pa ic it ol dr
AU13                                 27566  4
OU16                                        rb rs cf -d ar qr xl
CU13        52 fld    dl   d   415    20000  4 pa ol dr
AU14                                 20000  4
OU17                                        dl rs of -d ar qr
CU14        53 epp4   n*   i   415       54  4 pi pa ri ic wi it
AU15                          7  6   336054  4 ap sm pm
CU15             "    n*   n   234    27524  4 pa ri ic wi it cl pb
AU16                         12  5   533524  4 ap sm pm
CU16             "        d   255    43742  4 pa ic cl dr
AU17                                 43742  4
CU17        54 epp2   n*   n   255    43752  4 pa ri it cl
AU20                          4  4  3225752  4 ap sm pm
```

OUTPUT PRODUCED BY THE .HRLGD REQUEST

The .hrlgd request produces a list of the abbreviations used with the .hranl request above.

```
Abbreviations used in History Register Analysis

_____________CU Legend_____________
cy = cycle type (d = direct operand)
(i=instr. fetch,o=operand,F=fault)
(n=indirect,x=xec,*=nop,e=EIS)
mc = memory command
(00=rrs,sp; 04=rrs,dp; 10=rcl,sp)
(12=rmsk,sp; 16=rmsk,dp; 20=cwr,sp)
(24=cwr,dp; 32=smsk,sp; 36=smsk,dp)
(40=rd/lck; 54=rgr; 56=sgr)
(60=wrt/ulck; 62=con; 66=xec; 72=sxc)
>>>flags<<<
-y = memory address invalid
br = BAR mode
cl = control unit load
cs = control unit store
dr = direct operand
fa = prepare fault address
ic = IC value is odd
it = AR/PR reference
in = inhibited instruction
ol = operations unit load
os = operations unit store
pa = prepare operand address
pb = port busy or data from cache
pi = prepare instruction address
pl = port select logic not busy
pn = prepare final indirect address
pt = prepare operand tally
ra = request alter word
ri = request indirect word
rp = executing repeat
sa = store alter word
si = store indirect word
tr = transfer condition met
wi = request instruction fetch
xa = prepare execute interrupt address
xe = execute double from even ICT
xi = execute interrupt present
xo = execute double from odd ICT

_____________OU Legend_____________
>>flags<<<
9b = 9-bit byte (IT modifier only)
ar = A-register in use
d1 = first divide cycle
d2 = second divide cycle
dl = direct lower operand
du = direct upper operand
in = first ou cycle
it = IT character modifier
oa = mantissa alignment cycle
oe = exponent compare cycle
of = final OU cycle
om = general OU cycle
on = normalize cycle
os = second cycle of multiple ops
qr = Q-register in use
rb = opcode buffer loaded
rp = primary register loaded
rs = secondary register loaded
sd = store data available
-d = data not available
x0 = index 0 in use
x1 = index 1 in use
x2 = index 2 in use
x3 = index 3 in use
x4 = index 4 in use
x5 = index 5 in use
x6 = index 6 in use
x7 = index 7 in use
```

```
_____________DU Legend_____________
mc = data mode (b,4,6,9,w)
offset = descriptor counter
>>>flags<<<
()a = prepare alignment count for
      numeric operand (1,2)
a() = load alpha operand (1,2)
al = adjust length
as = alpha store
bd = binary-decimal execution

bg = blanking gate
c0 = force stc0
cg = character operation
d() = descriptor active (1,2,3)
da = data available
db = decimal-binary execution
dd = decimal unit idle
di = decimal unit interrupted
dl = decimal unit load
ds = decimal unit store
ei = mid-instruction interrupt enabled
en = end instruction
es = end sequence
ff = floating result
fl = first data buffer load
fp = first pointer preparation
fs = end sequence
l() = load descriptor (1,2,3)
ld = length = direct
lf = end first pointer preparation
lv = level < word size
lx = length exhaust
l< = length < 128
mp = executing MOPs
n() = load numeric operand (1,2)
nd = need descriptor
ns = numeric store
op = operand available
pc = alpha packing cycle
pl = prepare operand length
pp = prepare operand pointer
r() = load rewrite register (1,2)
re = write-back partial word
rf = rounding
rl = rewrite register 1 loaded
rw = du=rd+wt control interlock
sa = select address register
sg = shift procedure

_____________APU Legend_____________
seg# = SDWAMR and PTWAMR numbers if
corresponding MATCH bits are set.
offset = final store address
mc = ring number (TSR.TRR)

>>>flags<<<
an = final address, non-paged
ap = final address, paged
f  = access violation or directed
     fault
fd = fetch descriptor segment PTW
fh = fault waiting
fs = fetch SDW
md = modify descriptor segment PTW
mp = modify PTW
p1 = fetch PTW
p2 = fetch PTW+1
pm = MATCH in PTWAM
sm = MATCH in SDWAM

-------DU Legend-------
xg = exponent network
xm = extended al,ql modifier
+g = add-substract execution
*g = multiply-divide execution
```

Name: mos_edac_summary

SYNTAX AS A COMMAND

```
mos_edac_summary {-control_args}
```

FUNCTION

scans the syserr log and summarizes mos edac activity in a brief report.

CONTROL ARGUMENTS

-day_limit N
sets a threshold of N days that a memory chip can fail before including it in the summary. The maximum value for N is 16.

-for T
specifies a relative time (such as "1 hour") used to compute the ending time from the starting time.

-from DT, -fm DT
specifies the date/time to start scanning the log.

-limit N
sets a threshold of N edac errors for a memory chip before including it in the summary.

-mem list
specifies a list of memories for which information is required (i.e., mem a b c).

-to D
specifies the date/time to stop scanning the log.

Notes

If -from DT is not specified, the scan starts with the earliest message in the syserr log. The ending time may be specified by using -for, or -to, but not both. If both are omitted, the scan terminates which the last message in the log. All dates and times must be in a format acceptable to convert_date_to_binary_ described in the Subroutines manual.

You must have re access to audit_gate_ and r access to the permanent_syserr_log segment to use this command.

Name: mpc_data_summary

SYNTAX AS A COMMAND

```
mpc_data_summary {list} {-control_args}
```

FUNCTION

scans the syserr log and summarizes the MPC statistics placed there by poll_mpc.

ARGUMENTS

list
is a list of MPC controller names that the data is to be summarized for (i.e., mspa mtpb urpa). The MPC controller names must be four characters long, and the first three characters must be msp, mtp, or urp. The default list is of all MPCs found in the log.

CONTROL ARGUMENTS

-all
reports all MPCs found in the syserr log.

-brief, -bf
reports only nonzero device statistics.

-expand
expands each syserr log entry that is used for the summary. This may cause much output.

-extend, -ext
extends the output file if it exists. The default is to overwrite the file.

-for T
computes the ending time from the starting time, where T is a relative time (such as 1hour or 1day).

-from DT, fm DT
starts scanning the log at the date/time given.

-long, lg
reports all device statistics. (Default)

-mpc list
displays MPC error data only.

-output_file {path}, -of {path}
directs output to the segment specified by path. If path is not given, a default segment is used in the working directory and named mpc_data_summary.output.

-short
formats output for devices with fewer than 132 columns. The default is based on output file type and can be used to override the file output default.

-to DT
stops scanning the log at the date/time given.

Name: patch_firmware

SYNTAX AS A COMMAND

```
patch_firmware path mem addr word1...word2...wordi
```

FUNCTION

patches a segment containing an image of a firmware module for an MPC.

ARGUMENTS

path
is the pathname of the segment containing the firmware.

mem
is the memory overlay to patch. This argument can be cs to patch the control store overlay, or rw to patch the read/write memory overlay.

addr
is the starting address to patch, in hexadecimal.

wordi
is a new MPC word, in hexadecimal. All wordi arguments must be in the range 0-FFFF. At least one wordi argument must be specified. Up to 16 words can be patched with one patch_firmware command.

NOTES

The patch_firmware command displays the old and new contents of each firmware word patched, as well as the checksum, before the patch is made. The user is then asked whether the patch is correct. The patch is not made unless you answer yes.

Firmware modules can be retrieved from the IFAD tape using the load_tandd_library command (described in the Online T&D manual). Normally, firmware modules are kept in the sequential file >system_library_tandd>tandd_deckfile.

Name: poll_fnp

SYNTAX AS A COMMAND

```
poll_fnp {fnp_list} {-control_args}
```

FUNCTION

initiates and controls automatic polling of FNPs. Polling consists of reading error statistics from the FNP memory and logging them in either the syserr log or a file. This command sets up timers and event call handlers within the process. Once initiated, FNP polling is performed periodically, independent of whatever else is going on in the process. This command is normally used by the initializer or a daemon.

ARGUMENTS

fnp_list
: is a list of the FNP names to be polled. If no names are listed, all FNPs are polled.

CONTROL ARGUMENTS

-log
: writes statistical information to the syserr log. This is the default. Access to the hphcs_ gate is required.

-output_file path, -of path
: writes statistical information to the segment specified by path. This control argument can be used in conjunction with -log.

-time N, -tm N
: specifies the polling interval in minutes. The default polling interval is 15 minutes.

-debug, db
: prints extra debugging information each time polling is performed.

The following control arguments modify the polling already in process and cannot be used on the initial invocation of the poll_fnp command.

-stop, -sp
: stops polling for the FNPs specified with the fnp_list argument. If no FNPs have been specified, polling of all FNPs is stopped. Polling continues to be scheduled periodically, even though no FNPs are being polled.

start, -sr
: resumes polling for the FNPs specified with the fnp_list argument. If no FNPs have been specified, polling of all FNPs is resumed. Note that the next polling does not occur immediately; it is performed during the next scheduled polling cycle.

-finish
schedules the last polling cycle immediately. Once this cycle completes, polling is disabled, and a new poll_fnp command is required to start it again. To stop polling without performing one last cycle, use both -stop and -finish.

NOTES

If polling of an individual FNP fails three consecutive times, polling of that FNP is stopped. If three consecutive scheduled polling cycles are missed because a previous cycle did not complete, an automatic finish operation is performed, and no further cycles are scheduled.

Polling of FNPs has no effect on the users of devices connected to the FNP.

Name: poll_mos_memory

SYNTAX AS A COMMAND

```
poll_mos_memory
```

FUNCTION

reads the maintenance register of each memory on the system and prints information about these registers on your terminal. In addition, if the maintenance register indicates that an EDAC error has occurred, it is logged in the syserr log.

NOTES

You must have re access to phcs_ to use this command.

This command should be used with care on systems that have core memories. Unless the TEST/NORMAL switch on the maintenance panel of the memory (not controller) is set to TEST, the result of reading the maintenance register is undefined, and spurious errors may be logged.

Name: poll_mpc

SYNTAX AS A COMMAND

```
poll_mpc {mpc_list} {-control_args}
```

FUNCTION

initiates and controls automatic polling of MPCs. Polling consists of reading statistics on device usage and errors from the MPC memory and logging it in either the syserr log or a file. If an error condition is detected, a message is entered in the syserr_log with a code of 3. This sounds the BOS console alarm and prints the message on the BOS console log on a normally configured system. This command sets up timers and event call handlers within the process. Once initiated, MPC polling is performed periodically, independent of whatever else is going on in the process. This command is used by the initializer or a daemon; Utility.SysDaemon is recommended.

ARGUMENTS

mpc_list
: is a list of the tape or disk MPC names to be polled. If no names are listed, all tape and disk controllers are polled.

-log
: writes statistical information to the syserr log. Access to the hphcs_ gate is required. (Default)

output_file path, -of path
: writes statistical information to the segment specified by path. This report is the same as the one generated by the -stat control argument of the dump_mpc command. This control argument can be used in conjunction with -log.

-time N, -tm N
: specifies the polling interval in minutes. The default polling interval is 15 minutes.

-debug, -db
: prints extra debugging information each time polling is performed.

The following control arguments modify the polling already in process and cannot be used on the initial invocation of the poll_mpc command.

-stop, -sp
: stops polling for the MPCs specified with the mpc_list argument. If no MPCs have been specified, polling of all MPCs is stopped. Polling continues to be scheduled periodically, even though no MPCs are being polled.

-start, -sr
resumes polling for the MPCs specified with the mpc_list argument. If no MPCs have been specified, polling of all MPCs is resumed. Note that the next polling does not occur immediately; it is performed during the next scheduled polling cycle.

-finish
schedules the last polling cycle immediately. Once this cycle completes, polling is disabled, and a new poll_mpc command is required to start it again. To stop polling without performing one last cycle, use both -stop and -finish.

NOTES

If polling of an individual MPC fails three consecutive times, either because it cannot be attached or because of I/O errors, polling of that MPC is stopped. If three consecutive scheduled polling cycles are missed because a previous cycle did not complete, an automatic finish operation is performed, and no further cycles are scheduled.

Polling of MPCs has no effect on the users of devices connected to the MPC.

Name: print_configuration_deck, pcd

SYNTAX AS A COMMAND

```
print_configuration_deck {card_names} {-control_args}
```

FUNCTION

displays the contents of the Multics configuration deck. The data is kept up-to-date by the reconfiguration commands and, hence, reflects the current configuration being used.

SYNTAX AS AN ACTIVE FUNCTION

```
[pcd {card_names} {-control_args}]
```

ARGUMENTS

card_names
are the names of the particular configuration cards to be displayed. Up to 32 card names can be specified. (See the MOH, for the names of the configuration cards.)

CONTROL ARGUMENTS

-brief, -bf
suppresses the error message when a requested card name is not found. (Default)

-exclude FIELD_SPECIFIERS, -ex FIELD_SPECIFIERS
excludes particular cards or card types from being displayed. One to 14 field specifiers can be supplied with each -exclude, and up to 16 -exclude control arguments can be specified. To be eligible for exclusion, a card must contain fields that match all field specifiers supplied with any -exclude argument.

-long, -lg
prints an error message when a requested card name is not found.

-match FIELD_SPECIFIERS
selects particular cards or card types to be displayed. One to 14 field specifiers can be supplied with each -match, and up to 16 -match control arguments can be specified. To be eligible for selection, a card must contain fields that match all field specifiers supplied with any -match argument.

-pathname PATH, -pn PATH
prints card(s) from the copy of the configuration deck at PATH, rather than the one for the running system.

NOTES

Field specifiers can consist of a complete card field or a partial field and an asterisk (*). An asterisk matches any part of any field. For example, the field specifier "dsk*" would match any card containing a field beginning with the characters "dsk". Specifiers for numeric fields can be given in octal or decimal, but if decimal they must contain a decimal point. Asterisks cannot be specified in numeric field specifiers. All numeric field specifiers are converted to decimal and matched against numeric card fields, which are also converted to decimal. Hence, the field specifier "1024." would match a card containing the octal field 2000, and the field specifier "1000" would match a card containing the decimal field 512.

Selection is performed as follows. If no card names are specified, all cards are eligible for selection. On the other hand, if any card names are supplied, only the cards matching those names are eligible; and if more than one card exists with a specified name, all such cards are displayed. If a nonexistent card is requested, and the -long control argument is specified, an error message is displayed.

If any -match arguments are supplied, those eligible cards are matched against all field specifiers of each -match argument group; however, at least one -match group must have all its field specifiers match some field on the card to make that card eligible. A similar algorithm is used for any -exclude argument groups. So, if a card is eligible, and if -exclude arguments are supplied, then at least one -exclude group must have all its field specifiers match some field on the card to make that card ineligible. If no match for a given card name or -match group is found in the config_deck, nothing is displayed for that name or group, and no error is displayed. If no arguments are present, the complete config_deck is displayed.

Note that all card names must be specified before the first -match or -exclude argument. Field specifiers following a -match or -exclude argument include all arguments until the next -match or -exclude argument.

When called as an active function, the selected cards are returned in quotes, separated by a single space.

No action is taken for misspelled arguments or valid arguments for which there are no corresponding configuration cards.

EXAMPLES

```
! pcd cpu
   cpu  a  7  168  80.  on
   cpu  b  6  168  80.  on
   cpu  c  5  168  80.  off

(For the configuration deck displayed above.)

! pcd cpu -match on
   cpu  a  7  168  80.  on
   cpu  b  6  168  80.  on

! pcd -match 16 -ex off -ex b
   cpu  a  7  168  80.  on
```

Name: print_syserr_log

SYNTAX AS A COMMAND

```
print_syserr_log {-control_args}
```

FUNCTION

prints selected portions of the syserr log.

CONTROL ARGUMENTS

-pathname path, -pn path
: where path is the pathname of the segment to be used. The default is to use the perm_syserr_log.

The following control arguments determine which portions of the log are printed. If none are given, the entire log is printed. They can be chosen from any of these three groups:

The following control arguments specify the range of the log to be scanned:

-from DT, -fm DT
where DT is a decimal integer or a date/time. This argument specifies the starting point of the scan. If DT is an integer, it represents a sequence number; otherwise, it represents a date and time.

-to DT
where DT is a decimal integer, or a date/time. This argument defines the ending point in the scan by sequence number or time.

-for DT, -next DT
where DT is a decimal integer or a date/time. If a date/time is used, it must be a relative time (such as "1 day") that specifies how far from the starting point to scan the log.

-last N
where N is a decimal integer. This argument specifies that the scan is to start N messages back from the end of the log.

The starting point is specified by either -from or -last, but not both. If both are omitted, the scan starts at the earliest recorded message. The ending time is specified by -to or -for(-next), but not both. If both are omitted, the scan ends with the most recent message in the log. Date/time arguments used with -from, -for, or -to must be in a format acceptable to convert_date_to_binary_, described in the Subroutines manual.

The following control arguments specify which messages in the range scanned are to be, or not to be, printed:

-match STR1 ... STRn
where STRi are strings to be matched against messages in the log. Any message that contains an STRi is a candidate to be printed.

-exclude STR1 ... STRn, -ex STR1 ... STRn
where STRi are strings that are matched against the log, as for -match. Any message that contains an STRi is not printed. (Therefore, any message that does not contain an STRi is a candidate to be printed.)

-action A1 ... An
where Ai are decimal integers in the range 0 to 9. If this argument is used, only messages with an action code specified by an Ai are candidates to be printed.

-class C1 ... Cn, -cl C1 ... Cn
where Ci are decimal integers in the range 0 to 24. If this argument is used, only messages with a sorting class specified by a Ci are candidates to be printed.

If none of these control arguments are used, all messages in the range are printed. If some of the above control arguments are used, only messages that pass all these tests are printed.

The following control arguments specify the format of the messages printed.

-no_header, -nhe
specifies that the header that contains the range of the log under consideration is not printed.

-expand
specifies that messages that have binary data will have that binary data interpreted. The format is generally dependent on the text of the message.

-octal
specifies that all messages that have binary data will have that binary data printed in octal.

-limits
specifies that the command is only to read the first and last messages in the log and print their times and sequence numbers. No other action is performed, regardless of what other control arguments are used.

-debug, -db
inhibits all expanding of "=" messages in the log. All messages are printed exactly as they appear in the log.

NOTES

You must have re access to audit_gate_ and r access to the permanent_syserr_log segment to use this command.

EXAMPLES

To print the entire log, type:

```
print_syserr_log
```

The command line:

```
print_syserr_log -match "parity fault" -ex SysDaemon -expand
```

scans the entire log and prints all messages containing the string "parity fault" that do not contain the string "SysDaemon". The binary data logged with these messages is also printed. The result of this command is to print all parity faults logged by other than SysDaemon processes.

To see the messages for a certain time period that contain a particular string, use a command line of the form:

```
print_syserr_log -from "2/1/78 0000.0" -to "2/1/78 2400.0"
  -match RCP:
```

The -to control argument in this example could have been replaced by -for "1 day". All messages logged on 2/1/78 containing the string "RCP:" are printed.

The command line:

```
print_syserr_log -last 500 -class 2 -nhe
```

scans the last 500 messages in the log and prints out any messages with a sorting class of 2. This example also suppresses the header.

Name: save_history_registers

SYNTAX AS A COMMAND

```
save_history_registers {state} {-control_args}
```

FUNCTION

allows a user to save processor history registers upon each occurrence of a signalable fault in the signalers stack frame. By default, the history registers are not saved, and the history register block in the signalers stack frame is set to all zeros.

ARGUMENTS

state
: can be either "on" or "off." If state is not specified, it is off.

CONTROL ARGUMENTS

-priv
: specifies manipulation of the per-system state by directing the state and -print arguments to operate on the per-system history register save switch, wired_hardcore_data$global_hregs. When set, this switch causes all processes to save their history registers upon each occurrence of a signalable fault in the signalers stack frame. If -priv is not specified, then the state and -print arguments operate on pds$save_history_regs, the per-process history register save switch of your process executing this command.

-print, -pr
: displays the current state of the history register save switch if it is present without the state argument; with this argument, the state of the switch is displayed before the new state is applied.

NOTES

When -priv is used, hphcs_ access is required.

Name: set_mos_polling_time

SYNTAX AS A COMMAND

```
set_mos_polling_time {N}
```

FUNCTION

sets the time interval used by the system for polling MOS memories to check for and log EDAC errors.

ARGUMENTS

N
: is a decimal integer representing the time in minutes between MOS memory polls. If omitted, the command prints the current polling interval. If N is 0, MOS memory polling is disabled.

NOTES

MOS memory polling is disabled when the system is initialized. This command must be used to enable it.

You must have re access to hphcs_ to use this command.

MOS memory polling should not be enabled on systems that have core memories unless the TEST/NORMAL switch on the maintenance panel of the memory (not controller) is set to TEST. If this switch is set to NORMAL, spurious errors may be logged for the memory.

Name: set_proc_required, sprq

SYNTAX AS A COMMAND

```
set_proc_required {tag1}...{tag2}...{tagi} {-control_args}
```

FUNCTION

restricts processes to run only on specified CPUs. It can be used to specify the set of CPUs on which the invoking process can be run and the default set of CPUs for all processes that have not requested specific CPUs.

ARGUMENTS

tagi
is the tag for one of the CPUs in the group being specified. It can be one of the letters a through h or A through H. If no tag is specified, the group is assumed to contain all CPUs (tags A through H). If -priv is given, then at least one tag is required.

CONTROL ARGUMENTS

-priv
indicates that the group of CPUs specified is to become the default group for processes that have not requested specific CPUs. If omitted, the group of CPUs specified applies only to the invoking process.

NOTES

If none of the CPUs specified are online, an error message is printed, and the command has no effect.

This command requires access to phhcs_. If the -priv control argument is specified, access to hphcs_ is needed.

EXAMPLES

The command line:

```
set_proc_required A B
```

restricts the requesting process to run only on CPUs "A" and "B."

The command line:

```
set_proc_required
```

allows the requesting process to run on any CPU that is online

The command line:

```
set_proc_required A B E -priv
```

restricts all processes that have not requested specific CPUs to run only on CPUs "A," "B," and "E."

The command line:

set_proc_required -priv

allows all processes that have not requested specific CPUs to run on any CPU that is online.

Name: test_cpu

SYNTAX AS A COMMAND

```
test_cpu {-control_args}
```

FUNCTION

checks the CPU hardware for problems that have existed on the processors. By running various tests invoked by this command, you can determine whether the given CPU has had specific problems fixed. This command is usually used with the set_proc_required command if the system being tested has multiple CPUs configured.

CONTROL ARGUMENTS

-from TEST NUMBER/NAME, -fm TEST NUMBER/NAME
: starts testing from the test identified by TEST NUMBER or NAME. The default is to start testing from test 1.

-to TEST NUMBER/NAME
: stops testing after the test identified by TEST NUMBER or NAME. The default is to run all tests.

-test_names
: lists valid test names and the associated test numbers.

-exclude TEST_LIST, -excl TEST_LIST
: excludes the tests identified by TEST_LIST, where TEST_LIST is either a set of test names or numbers, from the tests that are run.

-stop_on_failure, -sof
: stops testing when a test failure occurs. The default is to continue testing with the next test.

-long, -lg
: displays machine conditions and history registers from a test failure. The default is not to display them.

-history_regs, -hregs
: displays history registers when a test fails. The default is not to display them.

-machine_conditions, -mc
: displays machine conditions when a test fails. The default is not to display them.

-brief, -bf
: inhibits display of the test numbers. The default is to display the test number and name as each test begins execution.

-repeat COUNT, -rpt COUNT
: repeats the test sequence the number of times specified by COUNT. The default is to run the test set one time.

-cycle COUNT
cycles on each test case the number of times specified by COUNT. The default is to run each test once.

-select TEST_LIST, -sel TEST_LIST
executes only those tests specified by TEST_LIST, where TEST_LIST may be either a valid test number or a name. The default is to run all tests.

-select TESTNAME1... {TESTNAME2}... {TESTNAMEi},
-sel TESTNAME1... {TESTNAME2}... {TESTNAMEi} is the name of a diagnostic test. If no test names are given, all tests are run and any failures are noted. If more than one test name is given, a list of all the tests is printed. The tests are described briefly below. To find out the exact details of each test, see the test_cpu program.

DIAGNOSTIC TESTS

mlrstern
checks a failure in which the fill character is placed as the first character on a page. This test causes a MME1 fault if the hardware fails.

tmlr
tries several MLR instructions, in several working combinations, across a page boundary. Messages are printed for any failures.

csl_oob
checks a particular use of a CSL instruction where the first descriptor is 0. This test causes an out_of_bounds fault if the hardware fails, and a MME1 fault if it succeeds.

mvn
checks the use of an MVN instruction that moves a number to a shorter number. The first two characters are dropped when the hardware fails.

mvn_of1
checks the use of MVN to move the number 0. An overflow indicates that the hardware failed.

tct
checks a particular TCT use. The test causes an op_not_complete if the hardware fails, and a MME1 fault if it succeeds.

sreg
checks the use of an SREG instruction that occurs as the last instruction in a page. The test causes an op_not_complete if the hardware fails, and a MME1 fault if it succeeds.

csl_onc
checks a particular CSL use. The test causes an op_not_complete if the hardware fails, and a MME1 fault if it succeeds.

test_sc2
checks the use of the SC modifier interacting with page faults. A MME1 fault occurs if the hardware fails.

test_ci
checks the use of the CI modifier interacting with page faults. A MME1 fault occurs if the hardware fails.

rpd_test
checks a particular use of the RPD instruction as it interacts with the hardware. A MME1 fault occurs if the hardware fails.

mlr_test
checks the use of the MLR instruction across a bounds fault boundary. The bounds fault is followed by a segment fault and a page fault. A MME1 fault occurs if the hardware fails.

cls_test
checks the CSL instruction across a bound fault boundary. A MME1 fault occurs if the hardware fails.

cmpc
checks the CMPC instruction in a way that fails if a timer runout or connect fault occurs in midexecution when the hardware is failing. A MME1 fault occurs if the hardware fails.

bad_fill
checks the success of moving or comparing fill characters in the first two words of a page. Failure is indicated by a miscompare and a message to the user.

mpy_ofl
multiplies -2**35 by itself and checks for an overflow fault (which indicates failure).

test_xed
checks a particular indexed XED usage that fails if the first executed instruction is an APU-type instruction. Failure is indicated by a miscompare and a message to you.

cmpc7
checks a CMPC failure when both strings begin seven words from a page boundary and run into the next page. A MME1 fault occurs if the hardware fails.

extra_fill
checks the MLR instruction to see if extra fill characters are placed after a string when the string crosses a page boundary. A MME1 fault occurs if the hardware fails.

test_cmpc_fill
checks the fill mechanism of the CMPC instruction near a page boundary. A MME1 fault occurs if the hardware fails.

acv_restart
checks that machine conditions can successfully be restarted after an access violation fault that is caused by a reference to data via an EIS (MLR) instruction. Failure is indicated by successive no_write_permission conditions.

scm_tally
checks to see if the SCM instruction works with the tally runout indicator set correctly. The test calls a small alm program that uses an SCM instruction. Because the hardware fails erratically, the test is run 10 times to get a (limited) statistical sampling. Failure is indicated by a message to you indicating the number of times the SCM instruction failed.

mvt_nine_to_six
checks nine to six (ascii to bcd) conversion using the MVT instruction. A large ascii data segment is generated. Then a bcd segment is generated using non-EIS conversion. Three segments are then converted from ascii to bcd using the MVT instruction, and these segments are compared to the known good bcd segment. If any compare errors are detected, the contents of both segments are dumped in octal at the failing location.

mvt_six_to_nine
checks six to nine (bcd to ascii) conversion using the method described for the mvt_nine_to_six test above. If any compare errors are detected, the contents of both segments are dumped in octal at the failing location.

mvt_nine_to_four
checks 9-bit to 4-bit (decimal to packed decimal) conversion using the MVT instruction. A large segment of data, containing 9-bit characters of values 0 to 15 in a rotating pattern, is generated. Then a second segment is generated, converting the 9-bit characters into 4-bit characters using non-EIS conversion techniques. The 9-bit data segment is then converted to three 4-bit data segments using the MVT instruction and compared to the known good 4-bit data. If any discrepancies are found, the contents of both segments are dumped in octal at the failing location.

mvt_four_to_nine
checks 4-bit to 9-bit (packed decimal to decimal) conversion using the method described for the mvt_nine_to_four test above. If any compare errors are found, the contents of both segments are dumped in octal at the failing location.

mvt_ascii_to_ebcdic
checks nine to nine (ascii to ebcdic) character conversion using the method described for the mvt_nine_to_four test above. If any discrepancies are found, the contents of both segments are dumped at the failing location.

mvt_ebcdic_to_ascii
checks nine to nine (ebcdic to ascii) character conversion using the method described for the mvt_nine_to_four test above. If any discrepancies are found, the contents of both segments are dumped in octal at the failing location.

ci_mod_case_2
checks character indirect modification with two tally words and two data character strings, each located at a page boundary. An LDA instruction is executed on one tally word, CI mod, and a CMPA is executed with a second tally word, CI mod. Both tally words point to a character string that should be equal. If the zero indicator does not come on as a result of the CMPA, a MME1 fault is taken, indicating that the hardware failed.

acv_restart_csl
validates that machine conditions can be successfully restarted after an access violation fault that is caused by a reference to data via an EIS (CSL) instruction. Failure is indicated by successive no_write_permission conditions.

cmpn_tst
checks that numeric data moved with an MVN instruction can be successfully compared with a CMPN instruction. Failure is indicated by a MME1 fault.

itp_mod
checks that an EPP2,* to a word pair that contains an ITP modifier with a bit offset actually loads PR2 with the correct information. A MME1 fault indicates failure.

mvnoosb
checks the prepage logic of the CPU for EIS numeric instructions. Failure is indicated by a MME1 fault.

cmpb_with_sixbit_offset
checks the CMPB instruction with a six bit offset. A MME1 fault indicates that the hardware failed.

cmpb_with_rotate
checks the CMPB instruction with a rotating pattern. A MME1 fault indicates that the hardware failed.

cmpc_pgbnd
compares a 38-character data string against a zero-length string. for a CMPC instruction that is located at seg|1767. Either an out_of_bounds condition or a MME1 fault indicates that the hardware failed.

csl_pgflt
checks that a CSL instruction does not get a no_write_perm condition if it causes a page fault on the target string and the source string is read-only.

scm_pgflt
tests a problem with the SCM instruction whereby the target operand takes a page fault and the resulting comparison is not made. Failure is indicated by a message to you indicating the number of miscompares.

scd_con_flt
tests a failure with the SCD instruction that fails when interrupted by a connect fault. Failure is indicated by displaying the number of times the SCD failed.

Name: test_dcw

SYNTAX AS A COMMAND

```
test_dcw {device} {name} {-control_args}
```

FUNCTION

constructs and executes arbitrary DCW lists on any device supported by the I/O interfacer.

ARGUMENTS

device
is the name of the device to be used. This can be either a specific device name, such as "tape_02" or "puna," or a generic device type, such as "printer" or "disk." If the device name is omitted, "tape" is assumed.

name
is the name of the tape or disk volume to be mounted. This argument is only used if the device is a tape or a disk, and is the name of the volume the operator is requested to mount. If the tape or disk volume name is omitted, "scratch" is assumed.

CONTROL ARGUMENTS

-read
places the device in read-only mode. This control argument only applies if the device is a disk or a tape.

-7track, -7tr
specifies a 7-track tape drive. This argument only applies if the device is a tape.

-priv
specifies a privileged attachment (see "Device Attachment" below.)

-sys
sets the system_flag in the rcp_ info structure during attachment (see "Device Attachment" below).

-debug, -db
runs the program in debug mode. In this mode, only the editing requests are recognized; no execution is allowed, and no actual device attachment takes place.

DEVICE ATTACHMENT

The test_dcw command attaches the device selected using the rcp_ subroutine. Normally, the call is made to rcp_$attach as a nonsystem process. However, if -priv is used, the call is made to rcp_priv_$attach. In both cases, if -sys is used, the system_flag in the rcp_ info structure is set, to indicate to rcp_ that you are to be considered a system process. You must have re access to the rcp_sys_ gate to make this kind of attachment. If the device specified in the command line is a device type rather than a specific device, rcp_ is relied upon to select the actual device to be used. In either case, the name of the device actually attached is printed after attachment completes.

COMMANDS

After the test_dcw command is invoked, commands are read from the user_input I/O switch. The following commands are recognized:

tdcw
: constructs a transfer DCW

idcw
: constructs an instruction DCW

nidcw
: constructs a nondata transfer IDCW

iotp
: constructs an I/O transfer and proceed DCW

iotd
: constructs an I/O transfer and disconnect DCW

iontp
: constructs an I/O transfer and proceed DCW

odcw
: constructs a DCW from octal input

pcw
: constructs a PCW

opcw
: constructs a PCW from octal input

edit, e
: selects a DCW list to edit

update, u
: places editor in "update" mode

insert, i
places editor in "insert" mode

delete, dl, d
deletes a DCW from the list

print, p
prints a DCW list

name
names a DCW list so that it can be referenced by name instead of number

save
saves all the current DCW lists in a segment

restore
restores DCW lists from a segment created by the save command

execute, x
executes a DCW list

getstat, g
checks for status from a previous operation

block, b
blocks process until an event occurs

xs
executes a DCW list, but leaves process blocked until special interrupt occurs

xr
executes a DCW list repeatedly, until some unusual status is returned

xre
executes a DCW list repeatedly, regardless of whether the operations succeed or fail

status, st
sets the current status reporting mode

rs
reprints the status from a previous operation

dump
dumps data from the I/O buffer on terminal

patch
inserts data into the I/O buffer from terminal

pattern
inserts data into the I/O buffer from the terminal by storing repeated copies of the data given

survey
displays data returned by a "survey devices" tape controller command

dtstat
displays data returned by a "read detailed status" tape handler command

chan
selects a specific IOM and channel for I/O

time
sets or prints the current time limit for ioi timeout

prompt
stores a character string to be used as a prompting message

susp
suspends I/O on devices connected to an MPC by calling ioi_$suspend_devices

rel
restores I/O on devices connected to an MPC by calling ioi_$release_devices

?
types out the current DCW list number, current DCW number, and the current editor mode

.
types the word "test_dcw" to verify that the test_dcw command is still in control

quit, q
releases attached device and returns

I/O BUFFER AREA

Once the device is attached, an I/O buffer is allocated using the ioi_ subroutine. The default length is 1024 words, although this can be changed later. The first 32 words of the buffer are reserved for DCW lists, and the second 32 words are reserved for the ioi_ status queue. When constructing a DCW list, care should be taken to avoid modifying the first 64 words (100 octal) of the buffer, or results (especially status reporting) may be unpredictable.

DCW LIST PREPARATION

The test_dcw command contains an editor that can create and update DCW lists using simple input statements. Up to 32 different DCW lists, each up to 32 words in length, can be created and selectively updated and executed. Each DCW list also has a PCW associated with it that, if present, is used instead of the system-supplied PCW when the list is executed. The 32 DCW lists are numbered from 1 to 32 in decimal. Each DCW list can also be given a name. The 32 DCWs in each list are numbered from 0 to 37 in octal.

The DCW editor keeps track of several quantities as DCWs are entered. These are the current list, the current DCW number, and the current mode. When the test_dcw command is invoked, the current list is 1, the current DCW is 0, and the mode is update.

When a DCW is entered in update mode, the new DCW replaces the current DCW in the current list, and the current DCW number is increased by one.

The editor can also be placed in insert mode. In this mode, when a new DCW is entered, all DCWs starting with the current DCW are shifted one position down the list, the new DCW replaces the position formerly occupied by the old current DCW, and the current DCW is increased by one. DCWs shifted out of position 37 octal are lost.

The edit command can be used to select a DCW list to edit, as follows:

```
edit {list} {name}
```

where:

1. list
   is either the name or number of the DCW list to edit. The list can also be specified as "*", in which case, the first available empty list is used.

2. name
   is the name given to the DCW list selected by the first argument. If omitted, the name of the list is not changed.

This command sets the current list to the one specified, the current DCW to 0, and the mode to update. If the list argument is omitted, the current list is not changed, but the current DCW and mode are set to 0 and update respectively.

A DCW list can be given a name (or a new name) with the name command.

```
name {name}
```

where:

1. name
   is the name to be placed on the current list. If omitted, the current list becomes unnamed. If some other list has the name specified, that list becomes unnamed.

The mode of the editor is controlled by the insert and update commands, as follows:

```
update {n}
insert {n}
```

where:

1. n
   is a DCW number, in octal. The update command puts the editor in update mode and sets the current DCW to n. Similarly, the insert command places the editor in insert mode. If n is omitted, the current DCW is not changed.

A DCW can be deleted from the middle of the list with the delete command.

```
delete {n}
```

where:

1. n
   is a DCW number, in octal. DCW n is deleted by moving everything after it in the list up one position. If n is omitted, the current DCW is deleted. The current DCW number is not changed.

Any of the following commands can be used to create a DCW:

```
idcw
nidcw
tdcw
iotd
iotp
iontp
odcw
```

After a DCW is constructed with any of these commands, it is edited into the current list, in the current position, according to the current mode, as described above. In all of the DCW commands described below, all numeric quantities are entered in octal. Any of the parameters shown are optional, and if omitted, the corresponding DCW field is zero (except for the device address field that is set to the address of the device assigned).

To create an IDCW, the command is entered as follows:

```
idcw {di} {args}
```

where:

1. di
   is the value to be placed in the device instruction field.

2. args
   are used to set the remaining fields in the IDCW and can be selected from the following:

   da oo
   places the value oo in the device address field.

   ci oo
   places the value oo in the channel instruction field.

   ae oo
   places the value oo in the address extension field.

   t oo
   places the value oo in the tally field.

   ec
   sets the extension control bit (ec bit).

   cont
   sets the continue bit.

   mark
   sets the marker status bit.

A nondata transfer IDCW can be entered more easily using the nidcw command. It is identical in format to the idcw command, but the tally defaults to 01 and the channel instruction defaults to 02.

A transfer DCW is created as follows:

```
tdcw {addr} {args}
```

where:

1. addr
   is the value to be placed in the address field.

2. args
   are used to set the remaining bits in the TDCW and can be selected from the following:

ec
sets the extension change bit (ec).

res
sets the restricted bit.

rel
sets the relative mode bit.

IOTD, IOTP, and IONTP DCWs can be entered using the commands shown below.

```
iotd {addr} {tally} {cp}
iotp {addr} {tally} {cp}
iontp {addr} {tally} {cp}
```

where:

1. addr
   is the value to be placed in the address field.

2. tally
   is the value to be placed in the tally field.

3. cp
   is the value to be placed in the character position field.

Any arbitrary DCW can be entered using the dcw command.

```
odcw {word}
```

where:

1. word
   is the octal DCW to be used. If word is omitted, an all-zero (and invalid) DCW is created.

Each DCW list can have one PCW associated with it. The PCW can be entered with the following command:

```
pcw {di} {args}
```

where:

1. di
   is the value to be placed in the device instruction field.

2. args
   are any of the optional args listed under the idcw command, with the following additions:

   mask
   sets the mask bit.

   reset
   sets bits 21, 22, and 23 to form a reset PCW.

Any arbitrary PCW can be entered with the opcw command as follows:

```
opcw {word}
```

where:

1. word
   is the octal PCW to be used. If word is omitted, an all-zero PCW is created and the system-supplied PCW is used on subsequent executions of the DCW list.

The DCW list can be displayed at any time using the print command.

```
print {list}
```

where:

1. list
   is either the name or number of a DCW list. If list is omitted, the current list is displayed. If list is specified, the current list is set to that list, the current DCW is set to 0, and the mode is set to update. Instead of a list name, "all" can be used to indicate that all DCW lists are to be displayed, or "names" can be used to list the names of all DCW lists.

SAVING DCW LISTS

Once edited, a permanent copy of all the current DCW lists can be saved in a segment for later use by invoking the save command.

```
save path
```

where:

1. path
   is the pathname of the segment where the data is to be saved. The segment always has a suffix of "test_dcw", which is supplied automatically.

To restore the previously saved DCW lists,

restore path

where:

1. path
   is the name of the segment created by the save command. If the command was not invoked in debug mode, all IDCWs and PCWs are updated with the device address of the device currently assigned.

I/O BUFFER EDITING

Several commands are available to edit and display the contents of the I/O buffer. To enter data into the buffer, the patch command is used.

patch offset word1...word2...wordi

where:

1. offset
   is the octal offset in the buffer to be patched.

2. wordi
   is the value to be placed in word offset+i.

Offsets less than 100 octal should not normally be used, as this could interfere with the DCW list, or the status queue.

If a repeating pattern is desired, use the pattern command.

pattern offset repeats word1...word2...wordi

where:

1. offset
   is the octal offset in the buffer where the data is to start.

2. repeats
   is an octal number representing the number of times the data is to be repeated.

3. wordi
   are the data words to be repeated.

To display the contents of a buffer (in octal), use the dump command.

dump {offset} {length}

where:

1. offset
   is the offset in the buffer to be dumped. If omitted, 100 octal is assumed.

2. length
   is the number of words to dump, in octal. If omitted, 10 octal is assumed.

If the data consists of 8-bit bytes in binary mode (unaligned, 9 in each two words), the dump command can be used to dump them. The format is the same as the dump command, except that the data is displayed in binary, and the length is given in bytes, instead of words.

If the data to be displayed is the output of a survey devices command issued to a tape controller, a special command can be used to display the data in a more meaningful way.

survey {offset}

where:

1. offset
   is the location in the I/O buffer where the data has been stored. If the offset is omitted, 100 octal is assumed.

If the data to be displayed consists of the output of a read detailed status command issued to a tape handler, it can be displayed with

dtstat {offset}

where:

1. offset
   is the location in the buffer where the status has been stored. If omitted. 100 octal is used.

EXECUTING THE DCW LIST

Once the DCW list is constructed, it can be executed as follows:

execute {list}

where:

1. list
   is the name or number of the DCW list to execute. If omitted, the current list is executed. If specified, the current list is changed to that list, the current DCW is set to 0, and the mode is set to update. The current list is copied into the I/O buffer starting at 0, and ioi_$connect is called to connect to relative address 0. If the list executed has a PCW associated with it, a call is made to ioi_$connect_pcw instead. After the connect is made, the process becomes blocked until an interrupt occurs. The status of the interrupt is then printed. If the status indicates that the channel is still running, the process goes blocked again waiting for another interrupt. If the channel is not running, test_dcw is ready to accept another command after the status is displayed.

If the DCW list being executed generates a terminate interrupt and a special interrupt (such as loading a tape drive), the following command can be useful:

xs {list}

This command is identical to the execute command, except that the process goes blocked after displaying the status from each interrupt until a special interrupt occurs.

A DCW list can be executed repeatedly using the following command:

xr {list}

This command executes the DCW list specified without displaying any status until an error condition is detected. The final status is printed normally.

Another variation of this can be used when it is necessary to repeat the DCW list, even though it has errors.

xre {list}

executes the list specified repeatedly regardless of the status. To terminate this, it is necessary to quit and to use the Multics program_interrupt command, afterwards.

Two other commands are occasionally useful in executing a DCW list.

block, b
getstat, g

The block command causes the process to go blocked waiting for an interrupt to occur. When it occurs, the resulting status is printed and test_dcw is ready for another command. The getstat command checks to see if any status is available, and prints it if it has occurred. The getstat command does not cause the program to go blocked if no status is available.

Using the block command (or if a channel fails), it is possible to put the process in a state where it is waiting for an event that never occurs. If this happens, a quit followed by a Multics program_interrupt command can be used to return to the test_dcw input routine.

STATUS REPORTING

Status is normally reported when received by printing it on the terminal. Status can be reported in three modes, as follows:

1. brief, bf
   is the default mode. The status message consists of the interrupt level in decimal, two words of IOM status in octal, and the major and minor status fields in binary.

2. long, lg
   consists of all eight words of the ioi_ status queue entry, in octal.

3. edited, ed
   is an English-language interpretation of the status.

The status mode is initially set to brief, but this can be changed as follows:

```
status {mode}
```

where:

1. mode
   is one of the three status modes described above. If omitted, the current mode is printed.

The previous status can also be redisplayed using the reprint status command.

```
rs {mode}
```

where:

1. mode
   is one of the three modes described above. If omitted, edited mode is assumed.

OTHER COMMANDS

Several other commands exist that can be useful. To set the length of the ioi_ timeout interval, use the time command.

time {n}

where:

1. n
   is the time limit in decimal seconds. If n is omitted, the command prints the current limit.

To change the size of the I/O buffer,

work {n}

where:

1. n
   is the buffer length desired in decimal words. If n is omitted, the work command displays the current buffer length.

To select a specific IOM and channel for I/O, the chan command can be used.

chan {iom} {channel}

where:

1. channel
   is the IOM channel, in octal.

2. iom
   is the IOM selected.

If channel is specified, but IOM is omitted, the IOM is assumed to be 1. If both are omitted, both are set to 0, indicating that ioi_ should make its own selection. The test_dcw command must be invoked with the -priv control argument in order to use this feature.

To suspend I/O on devices to connect to an MPC,

susp

To restore I/O, use

rel

These commands call the appropriate ioi_ entry points to accomplish their task. They are valid only if test_dcw was invoked with the -priv control argument and the device is connected to an MPC.

To read the special device status stored by the previous operation,

get_special_status

To read the detailed device status stored by the previous operation, use

get_detail_status

If a prompt message is desired when test_dcw is ready for input, it can be supplied by you as follows:

prompt {chars}

where:

1. chars
   is the data to be used for prompting. If chars is omitted, no prompt message is used.

To exit from the test_dcw command,

quit, q

The I/O device currently attached is detached, and the program terminates.

Name: test_fnp

SYNTAX AS A COMMAND

```
test_fnp FNP_tag {-control_args}
```

FUNCTION

tests DN66xx FNPs with the FED-supplied FNP test programs.

ARGUMENTS

FNP_tag
is the tag of the FNP to be tested. This FNP must have been shut down or FDUMPed; it cannot be involved in testing by another process. Level 6 FNPs cannot be tested with this command.

CONTROL ARGUMENTS

-exec name
specifies the FNP executive to be run initially. The name can be either "BOS" or "IOS." The default is BOS.

-input_switch name, -isw name
specifies the I/O switch from which operator input is read. The default switch is user_input.

-message_switch name, -msw name
specifies the I/O switch to which messages intended for the T&D line printer are written. The default switch is user_output. The FNP T&D programs generate output of this form if its query "IS A PRINTER AVAILABLE?" is answered affirmatively.

-output_switch name, -osw name
specifies the I/O switch to which messages intended for the operator console are written. The default switch is user_output.

NOTES

The FNP type of the FNP selected for testing is obtained from information contained in the Channel Definition Table (>system_control_1>cdt). If the user does not have access to this data base, a user query is issued in the form:

TEST_FNP: What is the FNP Type of FNP TAG?
Anwser: DN6600, DN6670, DN355, or quit.

If the "quit" response in entered, control is returned to the current command processor.

Users should be familiar with the FED offline version of TST3BT. The test options, queries, and message diagnostics relevant to FNP testing are produced by the FNP test programs themselves. The documentation for the offline version of TST3BT running under the PAS2 EXEC, and the T&D documentation for the FNP tests, contain information on actual dialogue with this program; it is the same as the dialogue with the offline version.

The operator console of TST3BT is simulated by the Multics terminal controlling the process running test_fnp. By default, test output appears on the terminal, and responses are expected from the terminal. Normal Multics input line editing applies to all responses, and lowercase input is acceptable.

The response "quit" to any query of test_fnp, regardless of how it was generated, terminates the test session, releases the FNP, and returns to command level.

The REQUEST button of the operator console is simulated by striking the QUIT key and using the program_interrupt (pi) command to return to test_fnp. Normally, the REQUEST button causes an interrupt to be sent to the FNP directing the FNP executive to enter its request loop.

Access to the tandd_ gate is required. Access to >sc1>cdt is required to obtain the correct model number of the FNP. If you do not have access to the CDT, the default model number is DN6678.

The tests executed by test_fnp are sorted in the keyed sequential vfile_ >system_library_tandd>tandd_deck_file. These tests are loaded from the FE distributed "FNP binary deck tapes" by the load_tandd_library command (described in Multics Online T&D).

Name: test_tape

SYNTAX AS A COMMAND

```
test_tape {-control_args}
```

FUNCTION

tests a tape drive or tape reel.

CONTROL ARGUMENTS

-volume ID, -vol ID
specifies a tape by its volume identification number, which can have a maximum of nine characters. If -volume is not given, a default of "test_tape" is used.

-comment STR, -com STR
allows you to pass additional information about the requested volume mount to the operator.

-device STR, -dv STR
selects a specific tape unit; STR must be the complete device name. If this control argument is not given, the system finds a free tape unit (e.g., -device tapb_08). It is incompatible with -compare.

-compare STR, -comp STR
writes and then reads a tape on device STR1, and then automatically has the operator mount the tape on device STR2 and read the tape. The mounting and reading continues to device STRn. At least two devices must be specified. Only one device is attached at a time. The full device name (e.g., -comp tapa_05 tapa_07) must be used. This control argument cannot be used with -device.

-density N, -den N
indicates the tape density, where N can be either 6250, 1600, or 800. The default is 1600.

-track7, -tk7
specifies a 7-track tape drive as the test unit. The default is 9 track.

-wait N, -wt N
attempts to attach the device N times, after one-minute waits, if the device desired is being used by another process. If after N waits the device still cannot be attached, the program bypasses the device. The default for N is two times.

-count N, -ct N
indicates the number of records to be written or read, where N is a decimal integer. Each write operation creates one 1040 word physical record. If this control argument is not given, then the entire tape is written or read.

-no_data_compare, -ndc
disables comparison of the data read to a known pattern. This control argument is useful for verifying that a tape can be read without knowing what data is on the tape.

-random
fills the data buffers with a known random data pattern. It cannot be used with -pattern.

-pattern N, -ptrn N
specifies N as the word of octal data to fill the data buffers, where N can be a maximum of 12 octal digits. If fewer than 12 digits are given, the field is padded on the left with zeroes. If this control argument is not given, a pattern of 222222222222 is used. The -pattern control argument cannot be used with -random.

-write_read, -wr
identifies the mode of the test. The tape is written and the read pass is preformed. (Default)

-write, -w
identifies the mode of the test. The tape is written and the read pass is bypassed.

-read, -r
identifies the mode of the test. The tape is mounted without a write ring and the read-only pass is preformed.

-raw
displays raw hex detailed status with each error message in addition to an interpreted display.

NOTES

The test_tape command senses the End of Tape Mark (EOT) and stops even if the record count has not been exhausted. Typing test_tape with no control arguments has the same effect as:

```
test_tape -vol test-tape -den 1600 -ct 100000 -ptrn 222222222222 -wr
```

Listed below is a summary of the default control argument values.

-volume	(test-tape)	-count	(100000 {entire tape})
-comment	(NONE)	-ndc	(OFF)
-device	(one previously assigned, or a free device)	-random	(OFF)
		-pattern	(222222222222)
-compare	(OFF)	-write	(ON)
-density	(1600)	-read	(ON)
-track	(9)	-raw	(OFF)
-wait	(OFF)		

SECTION 3

MULTICS HEALS

DESCRIPTION OF HEALS

HEALS (Honeywell Error Analysis and Logging System) assists field engineering and operations personnel in monitoring the performance of the hardware and provides a record of hardware operation for diagnosing transient malfunctions, tracking performance of hardware modules, and predicting scheduled maintenance.

HEAL SREPORTS

HEALS reports are initiated by the heals_report command (described later in this section). The names of desired reports, the time period of the reports, and the pathname of the report file are specified by arguments to the command. The reports are

io_error report
: all I/O errors logged to syserr log by the ioi_, disk_control, dn355, and bulk_store_control subroutine. The entries are in syserr log time sequence and contain the full octal status return word.

sorted_io_error report
: the I/O errors of the io_error report orders by day and by device address (IOM number, channel number, and device number); grouping the errors for the convenience of maintenance personnel. Within a device address, entries are further ordered by power off, major status, sub status, initiate/terminate interrupt, device command, IOM status, and record count residue. The octal status word is replaced (to keep the format width to 72 columns) by additional details of tape and disk errors.

cpu_error report
: history register data and other pertinent data for op_not_complete, parity, command, startup, and shutdown faults.

mos_edac_error report
: the MOS EDAC error entries in syserr log.

media_io_error report
: similar in content to the sorted_io_error report except that the primary sort key is media volume name (e.g., tape reel number).

EXAMPLES OF REPORTS

Examples of the HEALS reports that result from invocation of the heals_report command are shown on the following pages. The media_io_error report is not shown--its format and content are similar to the sorted_io_error report. If a problem is detected in processing an entry for the io_error report or the sorted_io_error report, the problem is reported with a comment line in place of data in the report entry. If the system is reconfigured between the time of logging an error and the time of execution of a HEALS run, reassigned channels or device names different from those obtained from the configuration table are not known to the report generators. These are reported as "ch_unkn" or "dv_unkn". The configuration known to HEALS is printed preceding an io_error or sorted_io_error report. If a device address cannot be determined, it is assigned IOM number 0 and channel number 0 so that the entries are grouped at the beginning of the sorted_io_error report. The numbers assigned 0,0 flag the entries as having invalid addresses.

Each entry of a report contains the syserr log sequence number and log time so that entries can be cross-referenced to the original syserr log (see Sections 1 and 2) and the HEALS log, and between the io_error and sorted_io_error reports.

Examples of the various HEALS reports follow.

Channel Assignment Table

The configuration known to HEALS that is printed out prior to an io_error or sorted_io_error report is shown below.

```
CHANNEL ASSIGNMENT TABLE AT TIME OF HEALS RUN
RUN DATE: 08/15/77              RUN TIME: 1620.4,
SYSTEM_ID: MR6.0              SITE_ID:  Honeywell

IOM          CHNL          DEVICE          MODEL
NUM           NUM            NAME         NUMBER

  1            08            prtd           1600
  1            09            prta           1200
  1            10            rdra            301
  1            11            puna            300
  1            12            prtc            301
  1            14            rdrb            201
  1            15            punb            201
  1            16            opc
  1            17            355a
  1            18            tape            500
  1            24            dska            451
  1            25            dska            451
  1            26            dska            451
  1            27            dska            451
  1            28            dskb            451
  1            29            dskb            451
  1            30            dskb            451
  1            31            dskb            451
```

```
IO_ERROR_REPORT: 08/14/77  1619.8 TO 08/15/77  1619.8          PAGE  1

SYSERR     LOG          DEVICE          STATUS  TLY     TAPE_NO  STATUS_RETURN
------------   ---------------   ------
TIME    NUMBER    NAME I-CC-DD CM   MJ-SB-I         DISK_AD

DATE: 08/14/77                                                  DATE: 08/14/77

1725.4   34421   rdra 1-10-01 01   02-01-t      5   N/A        420140000000
1809.8   34427   prtd 1-08-01 34   03-10-t      2   N/A        431000000000
1809.9   34429   prtd 1-08-01 34   02-01-i      1   N/A        420102000000
1822.4   34440   prtd 1-08-01 34   03-04-t      1   N/A        430400000000
1834.6   34441   prtd 1-08-01 34   02-01-i      1   N/A        420102000000
1917.5   34447   tape 1-18-01 15   13-22-t      5   .          532200000000
1926.5   34457   tape 1-18-03 15   13-22-t      1   mc019      532200000000
1939.5   34458   tape 1-18-03 15   13-22-t      1   mc019      532200000000
1955.4   34482   tape 1-18-04 15   13-22-t      1   mc020      532200000000
2000.8   34490   tape 1-18-02 15   13-22-t      1   mc021      532200000000
2006.7   34491   tape 1-18-02 15   13-22-t      1   mc021      532200000000
2012.3   34499   tape 1-18-01 15   13-22-t      1   mc022      532200000000
2017.7   34504   tape 1-18-03 05   12-10-t      1   m2088      521000000000
2017.9   34505   tape 1-18-03 05   12-10-t      1   m2088      521000000000
2018.4   34508   tape 1-18-01 15   13-22-t      2   mc022      532200000000
2023.2   34516   tape 1-18-04 15   13-22-t      1   mc023      532200000000
2034.1   34519   tape 1-18-04 15   13-22-t      7   mc023      532200000000
2045.2   34527   tape 1-18-02 15   13-22-t      1   mc024      532200000000
2047.9   34528   tape 1-18-02 15   13-22-t      1   mc024      532200000000
2053.7   34536   tape 1-18-03 15   13-22-t      1   mc025      532200000000
2103.8   34549   tape 1-18-03 15   13-22-t      3   mc025      532200000000
2116.8   34557   tape 1-18-04 15   13-22-t      1   mc026      532200000000
2120.8   34571   tape 1-18-01 15   13-22-t      1   mb025      532200000000
2208.2   34582   tape 1-18-03 15   03-10-t      1   m2068      431000000000
2357.9   34586   tape 1-18-01 15   13-22-t      2   mb025      532200000000

DATE: 08/15/77                                                  DATE: 08/15/77

0700.3   34610   dska 1-26-02 31   02-20-t      1              422000000100
0700.3   34612   dska 1-26-02 31                     422456
0700.3   34614   dska 1-26-02 31   extended: (40 00 00 00 82 00 00 00 00)
0714.0   34617   tape 1-18-01 15   13-22-t      2   mb026      532200000000
0728.2   34626   tape 1-18-02 15   13-22-t      1   mb027      532200000000

END: IO_ERROR_REPORT
```

SORTED_IO_ERROR_REPORT: 08/14/77 1619.8 to 08/15/77 1619.8 PAGE 1

DEVICE I-CC-DD	STATUS NAME	TLY CM	TAPE_NO MJ-SB-I	DENS	RING	TR< DISK_AD	CYL	HEAD	SEC	SYSERR LOG TIME	NUMBER
DATE: 08/14/77										DATE: 08/14/77	
1-08-01	prtd	34	02-01-i	1		N/A				1809.9	34429
1-08-01	prtd	34	02-01-i	1		N/A				1834.6	34441
1-08-01	prtd	34	03-04-t	1		N/A				1822.4	34440
1-08-01	prtd	34	03-10-t	2		N/A				1809.8	34427
end: prtd errors											
1-10-01	rdra	01	02-01-t	5		N/A				1725.4	34421
end: rdra errors											
1-18-01	tape	15	13-22-t	5		.	.	.	.	1917.5	34447
1-18-01	tape	15	13-22-t	1		mc022	1600	ys	df	2012.3	34499
1-18-01	tape	15	13-22-t	2		mc022	1600	ys	df	2018.4	34508
1-18-01	tape	15	13-22-t	1		mb025	1600	ys	df	2120.8	34571
1-18-01	tape	15	13-22-t	2		mb025	1600	ys	df	2357.9	34586
1-18-02	tape	15	13-22-t	1		mc021	1600	ys	df	2000.8	34490
1-18-02	tape	15	13-22-t	1		mc021	1600	ys	df	2006.7	34491
1-18-02	tape	15	13-22-t	1		mc024	1600	ys	df	2045.2	34527
1-18-02	tape	15	13-22-t	1		mc024	1600	ys	df	2047.9	34528
1-18-03	tape	15	03-10-t	1		m2068	800	ys	df	2208.2	34582
1-18-03	tape	05	12-10-t	1		m2088	800	ys	9	2017.7	34504
1-18-03	tape	05	12-10-t	1		m2088	800	ys	9	2017.9	34505
1-18-03	tape	15	13-22-t	1		mc019	1600	ys	df	1926.5	34457
1-18-03	tape	15	13-22-t	1		mc019	1600	ys	df	1939.5	34458
1-18-03	tape	15	13-22-t	1		mc025	1600	ys	df	2053.7	34536
1-18-03	tape	15	13-22-t	3		mc025	1600	ys	df	2103.8	34549
1-18-04	tape	15	13-22-t	1		mc020	1600	ys	df	1955.4	34482
1-18-04	tape	15	13-22-t	1		mc023	1600	ys	df	2023.2	34516
1-18-04	tape	15	13-22-t	7		mc023	1600	ys	df	2034.1	34519
1-18-04	tape	15	13-22-t	1		mc026	1600	ys	df	2116.8	34557
end: tape errors											
DATE: 08/15/77										DATE: 08/15/77	
1-18-01	tape	15	13-22-t	2		mb026	1600	ys	df	0714.0	34617
1-18-02	tape	15	13-22-t	1		mb027	1600	ys	df	0728.2	34626
end: tape errors											
1-26-02	dska	31	02-20-t	1						0700.3	34610
1-26-02	dska	31				422456	555	16	16	0700.3	34612
1-26-02	dska	31	extended:	(40 00 00 00		82 00 00 00	00)			0700.3	34614

SORTED_IO_ERROR_REPORT (cont)

DEVICE	STATUS	TLY	TAPE_NO	DENS	RING	TRK	SYSERR	LOG
I-CC-DD NAME CM	MJ-SB-I		DISK_AD	CYL	HEAD	SEC	TIME	NUMBER
DATE: 08/13/77							DATE: 08/13/77	
1-17-07 tapa 00	02-04-i	1					0902.7	34690
1-17-07 tapa 00	02-04-i	1					1013.6	34929
1-17-07 tapa 00	02-04-i	1					1046.3	35044
1-16-05 tapa 00	03-10-t	6	.	.	.	.	0842.3	34641
1-16-05 tapa 00	03-10-t	1	.	.	.	.	0842.6	34643
1-16-05 tapa 00	03-10-t	3	.	.	.	.	0844.6	34645
1-16-05 tapa 00	03-40-t	3	.	.	.	.	0847.7	34647
1-16-05 tapa 00	03-40-t	1	.	.	.	.	0847.8	34650
1-16-05 tapa 00	03-40-t	1	.	.	.	.	0847.8	34652
1-17-05 tapa 00	03-10-t	1	.	.	.	.	0842.3	34642
1-17-05 tapa 00	03-10-t	1	.	.	.	.	0843.5	34644
1-17-05 tapa 00	03-40-t	1	.	.	.	.	0847.7	34649
1-17-05 tapa 00	03-40-t	1	.	.	.	.	0847.8	34651
1-26-01 dska 00	00-03-t	1	0001496	1	18	16	1126.7	35133
1-26-01 dska 00	00-03-t	1	0001507	1	18	27	1129.3	35145
1-28-11 dskb 34	00-01-t	1	0081464	107	3	24	1332.3	35465
1-26-07 dska 00	00-20-t	1	0120512	158	10	32	0925.9	34736
1-26-07 dska 00	00-02-t	1	0121272	159	10	32	0926.0	34737
1-24-07 dska 35	00-03-t	1	0402248	529	5	08	0954.2	34843
1-26-07 dska 00	00-20-t	1	0404640	532	8	00	0954.1	34841
1-26-07 dska 35	00-20-t	1	0405400	533	8	00	0954.2	34842
1-20-01 dskc 34	00-20-t	1	0444384	584	13	24	1053.1	35075
1-20-01 dskc 00	00-20-t	1	0445144	585	13	24	1053.0	35074
1-28-11 dskb 00	00-20-t	1	0592040	779	0	00	1331.8	35464
1-16-04 tapa 00	03-10-t	1	dp012	dflt	ys	df	1355.3	35516
1-16-04 tapa 00	03-10-t	5	dp012	dflt	ys	df	1358.5	35522
1-16-01 tapa 00	03-40-t	11	dp126	dflt	ys	df	0838.7	34630
1-16-03 tapa 00	03-10-t	1	dp127	dflt	ys	df	0839.6	34640
1-16-03 tapa 00	03-10-t	2	dp127	dflt	ys	df	0849.6	34660
1-16-03 tapa 00	03-40-t	2	dp127	dflt	ys	df	0849.6	34663

END: SORTED_IO_ERROR_REPORT

```
CPU_ERROR_REPORT:        from 08/12/77  1081.7       to 08/12/77  1300.0
HEALS RUN OF 08/19/77    1102.0 ON SYSTEM MR6.0
```

CU Legend

```
cy = cycle type (d = direct operand)
(i=instr. fetch,o=operand,F=fault)
(n=indirect,x=xec,*=nop,e=EIS)
mc = memory command
(00=rrs,sp; 04=rrs,dp; 10=rcl,sp)
(12=rmsk,sp; 16=rmsk,dp; 20=cwr,sp)
(24=cwr,dp; 32=smsk,sp; 36=smsk,dp)
(40=rd/lck; 54=rgr; 56=sgr)
(60=wrt/ulck; 62=con; 66=xec; 72=sxc)
>>>flags<<<
-y = memory address invalid
br = BAR mode
cl = control unit load
cs = control unit store
dr = direct operand
fa = prepare fault address
ic = IC value is odd
in = inhibited instruction
ol = operations unit load
os = operations unit store
pa = prepare operand address
pb = port busy or data from cache
pi = prepare instruction address
pl = port select logic not busy
pn = prepare final indirect address
pt = prepare operand tally
ra = request alter word
ri = request indirect word
rp = executing repeat
sa = store alter word
si = store indirect word
tr = transfer condition met
wi = request instruction fetch
xa = prepare execute interrupt address
xe = execute double from even ICT
xi = execute interrupt present
xo = execute double from odd ICT
```

OU Legend

```
>>flags<<<
9b = 9-bit byte (IT modifier only)
ar = A-register in use
d1 = first divide cycle
d2 = second divide cycle
dl = direct lower operand
du = direct upper operand
in = first ou cycle
it = IT character modifier
oa = mantissa alignment cycle
oe = exponent compare cycle
of = final OU cycle
om = general OU cycle
on = normalize cycle
os = second cycle of multiple ops
qr = Q-register in use
rb = opcode buffer loaded
rp = primary register loaded
rs = secondary register loaded
sd = store data available
-d = data not available
x0 = index 0 in use
x1 = index 1 in use
x2 = index 2 in use
x3 = index 3 in use
x4 = index 4 in use
x5 = index 5 in use
x6 = index 6 in use
x7 = index 7 in use
```

```
___________DU Legend_____________
mc = data mode (b,4,6,9,w)
offset = descriptor counter
>>>flags<<<
()a = prepare alignment count for
      numeric operand (1,2)
a() = load alpha operand (1,2)
al = adjust length
as = alpha store
bd = binary-decimal execution

bg = blanking gate
c0 = force stc0
cg = character operation
d() = descriptor active (1,2,3)
da = data available
db = decimal-binary execution
dd = decimal unit idle
di = decimal unit interrupted
dl = decimal unit load
ds = decimal unit store
ei = mid-instruction interrupt enabled
en = end instruction
es = end sequence
ff = floating result
fl = first data buffer load
fp = first pointer preparation
fs = end sequence
l() = load descriptor (1,2,3)
ld = length = direct
lf = end first pointer.preparation
lv = level < word size
lx = length exhaust
l< = length < 128
mp = executing MOPs
n() = load numeric operand (1,2)
nd = need descriptor
ns = numeric store
op = operand available
pc = alpha packing cycle
pl = prepare operand length
pp = prepare operand pointer
r() = load rewrite register (1,2)
re = write-back partial word
rf = rounding
rl = rewrite register 1 loaded
```

```
___________APU Legend_____________
seg# = SDWAMR and PTWAMR numbers if
corresponding MATCH bits are set.
offset = final store address
mc = ring number (TSR.TRR)

>>>flags<<<
an = final address, nonpaged
ap = final address, paged
f  = access violation or directed
     fault
fd = fetch descriptor segment PTW
fh = fault waiting
fs = fetch SDW
md = modify descriptor segment PTW
mp = modify PTW
p1 = fetch PTW
p2 = fetch PTW+1
pm = MATCH in PTWAM
sm = MATCH in SDWAM
```

CPU_ERROR_REPORT (cont)

```
rw = du=rd+wt control interlock
sa = select address register
sg = shift procedure
xg = exponent network
xm = extended al,ql modifier
+g = add-subtract execution
*g = multiply-divide execution


syserr sequence #33228, at 08/12/77 1238.7;
syserr_log text: op_not_complete fault on CPU B by
                 Initializer.SysDaemon.z.

scu_data:            000033570041 000000000027 400326000120 000000000000
                     000230000200 342000000005 000006757120 000006757120

pointer registers:   61|5070  61|5120  33|446  61|4720
                     15|1374  15|1374  61|4720  61|0

index registers:     003126 005070 001260 000000
                     000002 000030 000241 000200


a: 000000002000 q: 000446000000  exp: 000 timer: 000331342 ring_alarm: 0

eis_info:            000400000000 000400000000 004620252000 771077777707
                     000000002000 000077777670 004576002004 000077777734

fault register:      010400000000

NUM       OU registers                        CU registers
1         627000627100 137767003101           200107764000 000033050020
2         213000213100 123777013505           201037710100 000447050200
3         450000450300 177777013522           201137710000 000224050200
4         736000236340 113777003107           300007235120 005072050020
5         736000736100 133777003110           200007235000 004145042011
6         621000621100 136777003122           600137735000 000226042201
7         431210431100 123777003123           200127735000 000007050015
10        275210275500 127777010221           300007035120 005074050020
11        757000757300 177777010222           200007035000 000050042011
12        740000740300 175777010223           600137735000 000230050201
13        213000213100 123777012172           200127735000 000003050015
14        735000235340 107777000224           300007413120 001464050021
15        735000735100 127777000225           200007413005 002000550010
16        035000035500 127777000226           700137757120 000232044201
17        735000735100 127777000227           300127757120 005076050021
20        413000735240 023777000230           300125757120 005076050002
```

```
NUM       DU registers                                  AU registers
1         777757037717 744243410017                     0006140006144 023321450775
2         737757037737 744243410017                     0003360012000 001145460775
3         737757037737 744243410017                     0003360001020 000001470775
4         777757037737 744243410017                     0006120006144 023331740775
5         737757037717 744243410017                     0001440006450 005621500775
6         777757037737 744243410017                     0003360012000 001145500775
7         737757037737 744243410017                     0003360001020 000001430775
10        777757037737 744243410017                     0001530000000 001775740775
11        777757037717 744243410017                     0001522201000 023521720775
12        737757037737 744243410017                     0001520111000 000403640775
13        777757037737 744243410017                     0007150000000 001775740775
14        777757037737 744243410017                     0007142011100 023521620775
15        737757037717 744243410017                     0400400400400 040040040040
16        777757037737 744243410017                     0400400400400 040040040040
17        737757037737 744243410017                     0002244000043 006440000000
20        737757037737 744243410017                     0777774000043 000001000000

 HR                          c
id##   IC_____ opcd__ tag_ y seg#_ offset__ mc flags

CU 1           lprp4       o            33  4 pa ic -y cl
CU 2           tra         i           447  4 pa tr wi it
CU 3       447 tra         i           224  4 pa tr ic wi
CU 4       224 lda    n*   n          5072  4 pa ri -y it cl
CU 5             "         o          4145  4 pa -y ol pb
OU14                                        rp rs in of ar
CU 6       225 als         i    61     226  4 pi pa ic wi pb
AU 1                          1  2 2332145  0 ap sm pm
CU 7             "         d    33       7  4 pa ic wi -y ol dr pb
OU15                                        rs of -d ar
CU10       226 adla   n*   n    33    5074  4 pa ri -y it cl
CU11             "         o    33      50  4 pa -y ol pb
OU16                                        rb rs of -d ar
CU12       227 als         i    33     230  4 pi pa ic wi pb
AU 2                        0      114546  0 an sm
CU13             "         d    33       3  4 pa ic wi -y ol dr pb
OU17                                        rs of -d ar
CU14       230 rscr   n*   n    33    1464  4 pa ri -y it cl pb
CU15             "    al   o    33    2000 54 pa -y ol
CU16       231 staq   n*   i    33     232  4 pi pa ri ic wi it pb
AU 3                                   147  0
AU 4                          1  2 2333174  0 ap sm pm
CU17             "    n*   n    14    5076  4 pa ri ic wi -y it cl pb
CU20             "    n*   F    14    5076  4 pa ri ic wi -y fa it pl
OU20                                        rp in -d ar qr

END: CPU_ERROR_REPORT
```

```
MOS_EDAC_ERROR_REPORT:      from 08/01/77  1059.5      to 08/07/77  1059.5
HEALS RUN OF 8/19/77  1059.7 ON SYSTEM MR6.0

       LAST ERROR                TALLY    ERROR     SYSTEM CONTROLLER REGISTER
LOG_NUM DATE        TIME                   RATE
                                           /MIN

  21019 08/01/77  1435.7          1       5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  21589 08/02/77  1049.8          1       5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  21649 08/03/77  1709.8          1       5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  22193 08/04/77  1146.8          2       5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  22273 08/04/77  1256.8          2       5.00      000000000000 140737400001
     EDAC error on mem b store a. MOS, 4k chip, Error: board Q, chip A67

  22274 08/04/77  1256.8          1       5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  22428 08/04/77  1441.8          4       5.00      000000000000 140737400001
     EDAC error on mem b store a. MOS, 4k chip, Error: board Q, chip A67

  22549 08/04/77  1646.8          4       5.00      000000000000 140737400001
     EDAC error on mem b store a. MOS, 4k chip, Error: board Q, chip A67

  22661 08/04/77  1951.8          2       5.00      000000000000 140737400001
     EDAC error on mem b store a. MOS, 4k chip, Error: board Q, chip A67

  22730 08/05/77  0001.9          1       5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  23343 08/05/77  1606.2          1       5.00      000000000000 340077400001

     EDAC error on mem b store a. MOS, 4k chip, Error: board R, chip A78
  23573 08/06/77  0412.3          1       5.00      000000000000 000137400001
     EDAC error on mem c store a. MOS, 4k chip, Error: board Q, chip A17


END: MOS_EDAC_ERROR_REPORT
```

HEALS IMPLEMENTATION

The functions of an error analysis and logging system are:

1. Capturing and logging hardware data.
2. Sorting and analyzing the data.
3. Presenting the analyzed data in a series of reports.

The logging function is performed by the syserr mechanism to syserr log as described in Section 1. The other functions are performed by the facilities described in this section.

The syserr log contains a number of entries not needed for the HEALS reports, and the time interval of syserr log data is normally not as large as may be desired for HEALS error data analysis. Therefore, the syserr log entries of interest to HEALS are extracted from the syserr log and written to an independent segment named >system_control_1>heals_dir>heals_log (hereafter referred to as the HEALS log).

The update_heals_log, truncate_heals_log, and print_heals_message commands are provided to manage the HEALS log.

The heals_report command creates a report for the specified time intervals and appends it to the output file, which is created if none exists. The default pathname of the output file is heals_reports in the working directory. The HEALS log is not updated or otherwise changed by the heals_report command. If the latest syserr log entries are wanted in the reports, the heals_report command must be preceded by the update_heals log command.

The segment heals_log and a control data segment (heals_log_info) are contained in the directory >system_control_1>heals_dir. Management of the HEALS log is expected to be done by field engineering personnel.

HEALS USAGE

HEALS is for use on both routine reporting of hardware errors and for specific reports on demand.

All HEALS reports should be generated on a daily basis following a HEALS log update to maintain a continuous record of hardware errors and malfunctions. This HEALS activity should be triggered by a scheduled absentee process such as the administrative "crank."

Any time that specific reports are wanted for monitoring or diagnostic purposes, the heals_report command can be invoked at the terminal with the name of the specific report desired (e.g., heals_report!io_error). Similarly, update_heals_log can be invoked by a privileged user of HEALS.

HEALS INSTALLATION REQUIREMENTS

The directory >system_control_1>heals_dir. created by asu.ec (system accounting startup), must exist.

The heals_log segment is created by the first invocation of the update_heals_log command.

HEALS COMMANDS

The commands that can be invoked to produce the HEALS reports are described in the remainder of this section. Command descriptions are presented in alphabetical order.

Name: heals_report

SYNTAX AS A COMMAND

```
heals_report {report_names} {-control_args}
```

FUNCTION

produces reports of interest to site-support and field-engineering personnel. The reports are appended to a report file specified in the -output_file control argument or by default to the heals_reports segment in the working directory. The ASCII report segment can be displayed, perused by you on the terminal, or printed on a high-speed line printer.

ARGUMENTS

report_names
can be one or more names from the following list (see -all below):

io_error
selects the I/O error report.

sorted_io_error
selects the sorted I/O error report.

media_io_error
is similar to the sorted io error report except that the primary sort key is the media volume name (e.g., tape reel number).

cpu_error
selects the CPU error report.

mos_edac_error
selects the MOS EDAC error report.

CONTROL ARGUMENTS

-output_file path, -of path
puts the report file in the file specified by path.

-from DT, -fm DT
specifies the date and time after which errors are reported. If this argument is not given, the default value is the value of -to time minus 24 hours.

-to DT
specifies the date and time up to which errors are reported. If this argument is not given, the default value is the current date and time.

-all, -a
specifies that all reports are to be generated. This argument can be used instead of listing all report names.

NOTES

The dates specified after the -fm, -from, and -to control arguments must be acceptable to convert_date_to_binary_ (see the Subroutines manual).

You must have r access on >system_control_1>heals_dir>heals_log.

EXAMPLES

If the command line:

```
heals_report io_error -from 03/01/78 -to 03/02/78
```

is issued at 2:00 PM; an ASCII report segment named heals_reports suitable for printing is created in the current working directory, containing the I/O Error Report for the period from 2:00 PM, March 1, 1978 to 2:00 PM, March 2, 1978.

Name: print_heals_message

SYNTAX AS A COMMAND

```
print_heals_message {-control_args}
```

FUNCTION

is a tool to be used by administrators for the maintenance of the HEALS log (the segment named >system_control_1>heals_dir>heals_log). It allows the printing of all or selected messages currently in the log. It can also be used to delete bad records from the log as well as to print out parts of each logged record.

CONTROL ARGUMENTS

-time DT
: selects all messages that occurred after the specified time. If omitted, a value of 0 is assumed.

-update
: allows you to delete selected messages from the HEALS log if you have the appropriate access. (See "Notes" below.)

-match STR
: selects messages with text containing the match string.

NOTES

You must have rw access on >system_control_1>heals_dir>heals_log for the update function; otherwise, r access is sufficient.

The date/time following the -time control argument must be of a form acceptable to convert_date_to_binary_ described in the Subroutines manual.

The print_heals_message command opens the heals_log segment with a mode of keyed_sequential_update to allow messages to be deleted. If a message is selected by using either the -time or the -match control argument, you can issue the following requests:

quit,q
: discontinues message processing and returns to command level.

next
: selects the next message that meets the specified selection requirements.

delete
: deletes the current record.

data
: prints the octal data contained in the current record.

EXAMPLES

The command line:

```
print_heals_message  -time 01/01/78  -match ioi_interrupt
```

sends to the user_output I/O switch all messages that were received after 01/01/78 whose ASCII text contains the string "ioi_interrupt".

Name: truncate_heals_log

SYNTAX AS A COMMAND

```
truncate_heals_log N
    or
truncate_heals_log {-control_args}
```

FUNCTION

deletes records from >system_control_1>heals_dir>heals_log. It is used with the update_heals_log command.

ARGUMENTS

N
: is the number of days, counted back from the current time, for which messages are to remain in the HEALS log.

CONTROL ARGUMENTS

-from DT, -fm DT
starts deleting messages from the specified date/time. If this control argument is omitted, a clock value of 0 is assumed; that is, the truncate_heals_log command starts deleting messages from the beginning of the log.

-to DT
stops deleting messages from the specified date/time. If omitted, a clock value equal to the current time is assumed.

NOTES

You must have rw access to the heals_log and heals_log_info segments, both located in >system_control_1>heals_dir, in order to delete messages from the HEALS log.

The date/times following the control arguments must be in a form acceptable to convert_date_to_binary_ (see the Subroutines manual).

Name: update_heals_log

SYNTAX AS A COMMAND

```
update_heals_log
```

FUNCTION

copies messages of interest to HEALS from the syserr log file into the HEALS log. The messages copied are those new messages added to the syserr log since the last invocation of this command by any process.

NOTES

In order to update the log, the directory >system_control_1>heals_dir must already exist, and you must have access to system files as follows:

re to audit_gate and to phcs_
r to system_control_1>perm_syserr_log
rw to system_control_1>heals_dir>heals_log
rw to system_control_1>heals_dir>heals_log_info

If either the segment >system_control_1>heals_dir>heals_log or the segment >system_control_1>heals_dir>heals_log_info does not exist, it is created: in this case, you need sma access on >system_control_1>heals_dir. The heals_log_info segment contains information about the current heals_log segment.

CONTROL ARGUMENTS

-from DT, -fm DT
starts deleting messages from the specified date/time. If this control argument is omitted, a clock value of 0 is assumed; that is, the truncate_heals_log command starts deleting messages from the beginning of the log.

-to DT
stops deleting messages from the specified date/time. If omitted, a clock value equal to the current time is assumed.

NOTES

You must have rw access to the heals_log and heals_log_info segments, both located in >system_control_1>heals_dir, in order to delete messages from the HEALS log.

The date/times following the control arguments must be in a form acceptable to convert_date_to_binary_ (see the Subroutines manual).

Name: update_heals_log

SYNTAX AS A COMMAND

copies messages of interest to HEALS from the syserr log file into the HEALS log. The messages copied are those new messages added to the syserr log since the last invocation of this command by any process.

```
update_heals_log
```

NOTES

In order to update the log, the directory >system_control_1>heals_dir must already exist, and you must have access to system files as follows:

re to audit_gate and to phcs_
r to system_control_1>perm_syserr_log
rw to system_control_1>heals_dir>heals_log
rw to system_control_1>heals_dir>heals_log_info

If either the segment >system_control_1>heals_dir>heals_log or the segment >system_control_1>heals_dir>heals_log_info does not exist, it is created; in this case, you need sma access on >system_control_1>heals_dir. The heals_log_info segment contains information about the current heals_log segment.

INDEX

L

M

O

P

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	MULTICS SYSTEM DIAGNOSTIC AIDS	**ORDER NO.**	AR97-03
		DATED	DECEMBER 1983

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ **DATE** _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

FOLD ALONG LINE

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

FOLD ALONG LINE

Honeywell

Together, we can find the answers.

Honeywell

Honeywell Information Systems
U.S.A.: 200 Smith St., MS 486, Waltham, MA 02154
Canada: 155 Gordon Baker Rd., Willowdale, ON M2H 3N7
U.K.: Great West Rd., Brentford, Middlesex TW8 9DH **Italy**: 32 Via Pirelli, 20124 Milano
Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F. **Japan**: 2-2 Kanda Jimbo-cho Chiyoda-ku, Tokyo
Australia: 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia**: Mandarin Plaza, Tsimshatsui East, H.K.

39546, 7.5C184, Printed in U.S.A.

AR97-03