z/VM

Diagnosis Guide

version 5 release 2

GC24-6092-01

z/VM

Diagnosis Guide

version 5 release 2

GC24-6092-01

Note:
Before using this information and the product it supports, read the information under “Notices” on page 245.

Second Edition (December 2005)

This edition applies to version 5, release 2, modification 0 of IBM z/VM (product number 5741-A05) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces GC24-6092–00.

Contents

About This Book

This book provides diagnostic guidance information to help IBM® customers identify, report, solve, and collect information about problems in the z/VM® operating system.

Who Should Read This Book

This publication is intended for system programmers, system analysts, users who will do diagnosis of z/VM, and users collecting data for diagnosis.

What You Should Know before Reading This Book

This publication assumes that you understand the hardware controls and features of your installation. It also assumes that you can use assembler language and have experience with programming concepts and techniques.

Where to Find More Information

You can find more information about VM and diagnosis in the publications listed in the back of this book. See “Bibliography” on page 251.

Links to Other Online Books

If you are viewing the Adobe Portable Document Format (PDF) version of this book, it may contain links to other books. A link to another book is based on the name of the requested PDF file. The name of the PDF file for an IBM book is unique and identifies both the book and the edition. The book links provided in this book are for the editions (PDF names) that were current when the PDF file for this book was generated. However, newer editions of some books (with different PDF names) may exist. A link from this book to another book works only when a PDF file with the requested name resides in the same directory as this book.

How to Send Your Comments to IBM

IBM welcomes your comments. You can use any of the following methods:

- Complete and mail the Readers' Comments form (if one is provided at the back of this book) or send your comments to the following address:

  IBM Corporation
  Department 55JA, Mail Station P384
  2455 South Road
  Poughkeepsie, New York 12601-5400
  U.S.A.
- Send your comments by FAX:
  - United States and Canada: 1-845-432-9405
  - Other Countries: +1 845 432 9405
- Send your comments by electronic mail to one of the following addresses:
  - Internet: mhvrcfs@us.ibm.com
  - IBMLink™ (US customers only): IBMUSM10(MHVRCFS)

Be sure to include the following in your comment or note:

- Title and complete publication number of the book
- Page number, section title, or topic you are commenting on

If you would like a reply, be sure to also include your name, postal or email address, telephone number, or FAX number.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Summary of Changes

This book contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change (in that edition only). Some product changes identified in this summary may be provided through z/VM service by program temporary fixes (PTFs) for authorized program analysis reports (APARs).

GC24-6092-01, z/VM Version 5 Release 2

This edition supports the general availability of z/VM V5.2.

64-bit Exploitation

z/VM V5.2 exploits the 64-bit addressing capability of IBM z/Architecture, and CP now uses storage above 2 GB for a much broader set of operations. Previously, guest pages had to be moved below 2 GB for many reasons. For example, guest I/O buffers for both standard I/O and QDIO were moved below 2 GB when an I/O operation was initiated. Now I/O can be done using buffers anywhere in real storage, and QDIO structures can reside above 2 GB, as can most CP control blocks.

Prior to z/VM V5.2, storage addresses in the system execution space (the virtual address space in which CP executes) were identity mapped to real storage. Now only the CP nucleus and the prefix pages are identity mapped, and most of the system execution space (also known as host logical storage) is not identity mapped. CP tables and control blocks in the system execution space are now accessed by their host logical storage addresses. Host logical storage pages may be backed with frames in real storage, but backing frames are not necessarily contiguous. Frames below 2 GB are used only when architecturally required, freeing CP to exploit backing frames above 2 GB for most operations.

Support for 64-Bit Dump

The 64-Bit Dump support changes the various dump producers so that they create dumps that include relevant storage above 2 GB if that storage is defined. Prior to this release, all dump producers (Stand-Alone Dump, Hard Abend Dump, Soft Abend Dump and VMDUMP) dumped storage only below 2 GB.

Support for Vector Facility Removed

Because the Vector Facility feature is not available on any servers that z/VM supports, CP support for the Vector Facility has been removed.

Guest LAN Sniffing support

Tracing Guest LAN or VSWITCH paths also supported.

GC24-6092-00, z/VM Version 5 Release 1

This edition supports the general availability of z/VM V5.1.

SCSI FCP Disk Support

z/VM now supports SCSI FCP disk logical units (SCSI disks) for both system and guest use. As part of this support, the following trace table codes are added: 2890, 28A0, 2C90, 2CA0, 2CB0, 6000, 6001, 6002, 6003, 6004, 6005, 6006, 6010, 6011, 6012, 6013, 6014, 6015, and 6016.

Removal of CP Functions

z/VM V5R1 is designed to operate only on IBM zSeries, or equivalent servers that support IBM z/Architecture (64-bit). As a result, certain functions are not provided by z/VM V5R1:

- IPL from a 31-bit image of the CP nucleus
- Preferred (V=R and V=F) virtual machines
- Paging of the CP nucleus

For information about the affected external interfaces, see the *z/VM: Migration Guide*.

Chapter 1. Introduction to Debugging

z/VM manages the resources of a single computer such that multiple computing systems appear to exist. Each “virtual computing system”, or virtual machine, is the functional equivalent of a real processor. Therefore, the person trying to determine the cause of a z/VM software problem must consider these separate areas:

- The Control Program (CP), which controls the resources of the real machine
- The virtual machine operating system running under the control of CP, such as CMS (Conversational Monitor System) or GCS (Group Control System)
- The problem program that was running under control of the virtual machine operating system when the problem occurred.

See:

- Chapter 2, “Debugging Interactively,” on page 25 for information on how to debug problems within a virtual machine
- Chapter 5, “Debugging CP,” on page 55 for information on CP
- Chapter 6, “Debugging CF Service Machine Problems,” on page 67 for information on CF service machines
- Chapter 7, “Debugging CMS,” on page 69 for information on CMS
- Chapter 8, “Debugging CMS Pipelines,” on page 85 for information on CMS Pipelines
- Chapter 9, “Debugging the SFS Server or CRR Recovery Server,” on page 95 for information on the SFS and CRR server machines
- Chapter 10, “Debugging GCS,” on page 103 for information on GCS
- Chapter 11, “Debugging TSAF,” on page 181 for information on TSAF
- Chapter 12, “Debugging AVS,” on page 189 for information on AVS.

This guide contains information about using the Dump Viewing Facility and VM Dump Tool for debugging. For complete information explaining how to use the Dump Viewing Facility, see the *z/VM: Dump Viewing Facility* book. For complete information explaining how to use the VM Dump Tool, see the *z/VM: VM Dump Tool* book.

If a problem is caused by a guest operating system, see the publications pertaining to that operating system for specific information.

If it becomes necessary to apply a Program Temporary Fix (PTF) to a component of z/VM, see the *z/VM: Service Guide* for information on applying PTFs.

How to Start Debugging

A good approach to debugging is to:

1. Recognize that a problem exists
2. Identify the problem type and the area affected
3. Analyze the data you have available, collect more data if you need it, then isolate the data that pertains to your problem
4. Determine the cause of the problem and correct it, or report it to the appropriate IBM Support Center.

Does a Problem Exist?

The most common problems occurring on your z/VM system or virtual machine are:

- Abnormal end (abend)
- Unexpected or incorrect result
- Infinite loop
- Wait state
- Hang condition
- Slow performance.

Abnormal End

The most obvious indication of a problem is the abnormal end (abend) of a program. An abend occurs when an error condition that cannot be resolved by the system causes a program to end prematurely. Whenever a program abnormally ends, a message is issued. This message provides information that can help you isolate the problem. A dump often accompanies an abnormal end. See “Abends” on page 14 for a description of the various types of abends and their possible causes.

Unexpected or Incorrect Result

Another obvious indication of a problem is unexpected or incorrect output or result. If your output is missing, incorrect, or in a different format than expected, a problem exists. For more information, see “Unexpected Results” on page 16.

Infinite Loop

A loop is a set of instructions that are run repeatedly as long as one or more conditions are present. However, when the condition that is supposed to be satisfied in the loop is never reached, an infinite loop occurs. If your program takes longer to run than anticipated, or if your output is repeated more than expected, your program may be in an infinite loop. For a description of different types of loops, see “Loops” on page 17.

Wait State

A z/VM system or virtual machine is in a wait state between the time the system asks for data and begins to receive it. No other processing can occur in a system or virtual machine that is in a wait state. When the system or virtual machine is in a *disabled* wait state, it accepts no incoming data. When the system or virtual machine is in an *enabled* wait state, it continues to accept incoming data. Enabled wait states occur frequently and are quite easily resolved or resolve themselves. Disabled wait states are not easily resolved and almost always signal a serious problem, but often a message is issued alerting you to a disabled wait. If your program is taking longer than expected to run, the virtual machine may be in a wait state. See “Wait States” on page 19 for a closer look at the common types of wait states.

Hang Condition

A hang condition occurs when either CP cannot continue processing or a virtual machine cannot be dispatched. As a result, z/VM halts processing. For more information, see “Hang Conditions” on page 21.

Slow Performance

Your system is not limited to the problems listed above. Other problems that are not easily determined may appear to slow the system’s performance or cause unproductive processing time. These can be caused by poor system tuning or problems with your hardware. See the *z/VM: Performance* book for information about system tuning and performance.

Identifying the Problem

Identifying problems is not always easy. An abnormal end is indicated by an error message. Unexpected results become apparent after the output is examined. Loops, wait state conditions, and hang conditions may not be as easy to identify as an abend or unexpected results.

Table 1 summarizes problem types and the areas where they may occur.

Table 1. z/VM Problem Types

Problem Type	Where Problem Occurs	Distinguishing Characteristics
Abend	CP CF service machine CMS GCS TSAF AVS	For a complete discussion of reasons for abends and system programmer's actions, see the CMS, CP, GCS, TSAF, and AVS abend code charts in the appropriate component of the messages and codes books.
	Virtual machine abend (other than CMS)	When z/OS or VSE abnormally ends on a virtual machine, the messages issued and the dumps taken are the same as they would be if z/OS or VSE abnormally ended on a real machine. CP may stop a virtual machine if an irrecoverable machine check occurs in that virtual machine. The system operator receives a message at the processor console. Also, the virtual machine user is notified that his virtual machine was terminated.
Unexpected Results	CP	If an operating system runs properly on a real machine, but not properly with CP, a problem exists. Inaccurate data in files, such as spool files, is an error.
	Virtual machine	If a program runs properly under the control of a particular operating system on a real machine, but does not run correctly under the same operating system with CP, a problem exists.
Wait	CP	For a complete discussion of CP and loader wait state codes, see the *z/VM: CP Messages and Codes* book. See the *z/VM: System Operation* book for stand-alone dump codes.
Loop	CP disabled loop	The processor console wait light is off. The problem state bit of the real PSW is off. No I/O interrupts are accepted.
	Virtual machine disabled loop	The program is taking longer to run than anticipated. Signaling attention from the disabled loop terminal does not cause an interrupt in the virtual machine. The virtual machine operator cannot communicate with the virtual machine's operating system by signaling attention.
	Virtual machine enabled loop	Excessive processing time is often an indication of a loop. Use the CP QUERY TIME command to check the elapsed processing time. If time has elapsed, periodically display the virtual PSW and check the instruction address. If the same instruction, or series of instructions, continues to appear in the PSW, a loop probably exists.

Table 1. z/VM Problem Types (continued)

Problem Type	Where Problem Occurs	Distinguishing Characteristics
Performance	System hang	z/VM cannot complete any tasks. No I/O interrupts are accepted.
	User hang	The program is taking longer to run than anticipated. No I/O interrupts are accepted.

Immediate signs of problems within a user's virtual machine are:

- Return codes
- Error messages.

Return Codes

A return code is a number generated by the software associated with a computer program. This return code indicates to your program the condition that arose when your machine tried to carry out the program. Based on this condition, the return code influences your program in determining how subsequent processing of your overall task should proceed.

You must design your program to respond to specific return codes in specific ways. Your z/VM system—its system programming—is no different. Depending upon the return code received from a program in its system software (or, for that matter, in an application program that you are running on z/VM), your system is programmed to react in a certain way.

Return codes differ in severity. Some conditions are handled more smoothly than others.

For an explanation of the meaning of individual return codes, see the appropriate component of the messages and codes books.

Messages

A message is a sentence or phrase transmitted by z/VM that describes a situation or problem the system encountered while processing an instruction or command. Like a return code, it describes a situation and influences a reaction to it. Unlike a return code, which is generated for the benefit of a running computer program, a message is issued for the benefit of the person who wrote the program or issued the command.

z/VM has many messages and is programmed to generate a particular one when a given situation or problem occurs.

Messages consist of a message identifier (for example, DMSACC017E) and message text. The identifier distinguishes one message from another. The text is a phrase or sentence which either describes a condition that has occurred, or requests a response from the user.

For an explanation of individual messages, see the appropriate component of the messages and codes books.

Analyzing the Available Data

Sources that are available to help identify and correct a problem include but are not limited to:

- A dump

- A nucleus load map (NUCMAP)
- Registers
- The program status word (PSW)
- The console log
- A trace
- The symptom record.

You may need to use one or more of the above sources, or others, to find exactly where a problem occurred. For an explanation of individual abend codes, see the appropriate component of the messages and codes books.

Dump

A dump is a record of the contents of your machine's storage at a given moment.

For more information on dumps and how to use them, see Chapter 4, "Creating a Dump," on page 49 and Chapter 5, "Debugging CP," on page 55.

Nucleus Load Map

A nucleus load map (NUCMAP or load map) is a file that contains the following information:

- A list of the storage addresses of all control sections (CSECTs). A control section is the part of a program that the programmer defines as a relocatable unit. It is a block of code that can function properly in any part of storage. All elements of a CSECT are loaded into adjoining locations in storage.
- The storage addresses of all modules loaded into the CP nucleus, CMS nucleus, or GCS nucleus. The CP nucleus contains that portion of CP resident in host storage. Similarly, the CMS or GCS nucleus is that portion of CMS or GCS present in virtual storage.
- A list of all modifications performed on the modules in the nuclei. This includes all the maintenance that IBM has performed on the modules and all the modifications your organization has made to them.

One load map exists for CP, another for CMS, and another for GCS. z/VM creates a load map each time CP or CMS is built—that is, when your system is first installed or after it is repaired or modified. [1] In this manner the load maps are kept up to date.

Load maps are useful particularly when you are dealing with an infinite loop. Load maps also complement the information found in a dump. When you use one, you should have the other handy.

Load maps can be found in the following locations:

- The CPNUC MAP file, on the MAINT virtual machine's disk at virtual address 194, contains the CP NUCMAP.
- The CMSNUC MAP file, on MAINT's disk at virtual address 193, contains the CMS NUCMAP.

1. These activities are performed by the system programmer or system operator using the MAINT virtual machine. This is the virtual machine you use to install, service, and maintain your z/VM system. The *z/VM: Guide for Automated Installation and Service* and *z/VM: Service Guide* explain these activities.

Registers

A register is an area of storage specially set aside in your processor. Your machine is equipped with a prefix register, 16 general purpose registers, 16 control registers, 16 access registers, and 16 floating-point registers.

General purpose registers contain information being manipulated by the user program currently running. Floating-point registers hold numeric values associated with some exponent. These are usually very small or very large numbers—for example, 45.6×10^{12}. While general and floating-point registers contain data directly related to the processing of a user application program, control registers are used to calculate and keep track of certain values pertaining to the operation and management of the z/VM system. Access registers can designate any address space, including the current instruction space.

Your program uses a register to store a piece of data that it is currently using. A register can contain a numeric or alphabetic value, an address, or an instruction that the computer is currently using to do some small step in your overall task.

A register holds a piece of data only as long as it is needed. The traffic in and out of any given register can be quite heavy. A great deal can be learned by examining the contents of your system's registers if a problem occurs.

The contents of your system's registers are included in any dump. It is also possible to examine the contents of your registers by issuing various commands and during a trace.

Program Status Word

The PSW (program status word) indicates your system's general status. There are six different types of system PSWs that provide diagnostic information. Each one has an old and new value. The PSWs are as follows:

- Restart
- External (EXT)
- Supervisor call (SVC)
- Program (PGM)
- Machine-check (MCH)
- Input/output (I/O).

The PSW format is described in detail in the *ESA/390 Principles of Operation* or *z/Architecture Principles of Operation*. The state of your system, whether it is waiting or processing, whether it can receive I/O interrupts or not, and the address of the next instruction to be executed are reflected in these parts of the PSW:

Bit 6 Indicates whether your system accepts (or is enabled for) input/output (I/O) interrupts. If this bit is set to 0, your machine is not enabled for I/O. If this bit is set to 1, your machine accepts I/O interrupts.

Bit 12 Indicates the architecture mode. It is 1 when in ESA/390 mode and 0 when in z/Architecture mode.

Bit 14 Indicates whether your z/VM system is in a wait state. If this bit is set to 0, your system is not in the wait state, and processing can proceed normally. If this bit is set to 1, your system is in a wait state.

If bit 14 is set to 1, the setting of bit 6 usually[2] indicates whether the wait state is enabled (1) or disabled (0).

Bits 64–127 (for z/Architecture mode) or 33–63 (for ESA/390 mode)
Contain the address of the next instruction your machine is set to process.

Examining the current PSW periodically may help you identify a loop. If the PSW instruction address always has the same value, or if the instruction address has a series of repeating values, the program probably is looping.

You can see the contents of the PSW by using the CP DISPLAY command with the PSWG option for z/Architecture (64-bit) mode or the PSW option for ESA/390 (31-bit) mode. You can also determine the PSW by looking at a dump.

Console Log

A console log is a record of everything that has appeared on a certain virtual machine's console. This includes all commands, messages, return codes, and results.

When problems arise in the system, you are generally interested in the console log for the system operator's console. The log includes all:

- Messages and return codes that have been sent to the operator
- Commands and instructions entered at the operator's console
- Responses that the operator has made to requests for action by the system.

The console log can describe the sequence of events that lead to a malfunction, error, or problem from the system's point of view.

It is not always just the system operator's console log that might help you. For example, if you are having a problem with RSCS, the console log for the RSCS virtual machine might help.

At the system operator's console, the recording of the console log is automatic and takes place at all times. To get a console log at other consoles you must enter the command:

```
cp spool console start
```

to begin the recording. The best place for this command for CMS users is in the PROFILE EXEC for the virtual machine in question, or in the PROFILE GCS for GCS users. That way, you know a console log is always being recorded. You can also enter the command from the command line and have it in effect temporarily.

Issue

```
cp spool console close
```

to create a console log of the information recorded up to this point and put the file in your virtual printer. Recording continues until you log off the system or explicitly stop it with the CP SPOOL CONSOLE STOP command.

To close and purge the spool file of an existing log, issue:

```
cp spool console purge
```

2. Bit 6 can be set to 0 and bit 14 set to 1 without the CPU being in a disabled wait state. For example, this could occur if bit 7 is on and the program is waiting for an external interrupt.

Traces

A trace is a chronological record of every major event that has taken place within your z/VM system or within a virtual machine running there. A major event corresponds to a program or set of instructions that your system or virtual machine has run, representing a major accomplishment in an overall task. The trace shows how each event affected virtual storage, registers, the PSW, and other aspects of your system.

A trace is invaluable when trying to track down a problem, particularly in the case of wait states, infinite loops, and unexpected output. Often, traces themselves suggest solutions to the problem. In a trace, you see the overall effect of every event that occurred before and after the problem arose.

When CP tracing is active in z/VM, system events are recorded as trace table entries in real storage. The number of trace table pages available to a processor is determined by the TRACE portion of the STORAGE statement in the system configuration file. You can override the effect of the TRACE portion of the STORAGE statement by using the CP SET TRACEFRAMES command. The trace table is described in the section titled “Trace Entries” on page 39.

An internal trace table is maintained for GCS. Consult “Internal Tracing Facilities” on page 103 for more information.

z/VM and GCS provide several commands you can enter to generate a trace of your own. Each has certain characteristics that appeal to certain needs, as explained below.

TRACE
: A CP command that monitors events in a virtual machine. The TRACE command monitors such events as instruction processing, I/O activity, successful branching, or a change in a register or storage location. This command records trace data in a trace entry that you can send to a virtual console, a virtual printer, or both. For more information, see the *z/VM: CP Commands and Utilities Reference*. Also, review the section of this book titled “Commands That Monitor Events” on page 29.

TRSAVE
: A CP command that saves trace data. You can save CP trace table data in system trace files or on tape. You can save trace data defined by the TRSOURCE command in system trace files only. For more information, see Chapter 3, “Using Traces to Debug,” on page 39 and the *z/VM: CP Commands and Utilities Reference*.

TRSOURCE
: A CP command that defines a trace as an I/O trace (IO), a data trace (DATA), or a guest trace (GT). TRSOURCE also activates or deactivates a trace, displays the status of a trace, and removes trace IDs from CP. For more information, see Chapter 3, “Using Traces to Debug,” on page 39 and the *z/VM: CP Commands and Utilities Reference*.

ETRACE
: A GCS command that initiates the recording of events. The ETRACE command works with the TRSOURCE command. For more information, see Chapter 10, “Debugging GCS,” on page 103 and the *z/VM: Group Control System* book.

ITRACE
: A GCS command that enables or disables the recording of events in the

GCS internal trace table. Rather than record events taking place in the system as a whole, the GCS internal trace table records events within a virtual machine or virtual machine group. For more information, see “Using the ITRACE Command and GTRACE Macro” on page 104 and the *z/VM: Group Control System* book.

There are even more tracing tools for those interested in the Systems Network Architecture (SNA). VTAM® and NCP provide SNA users with several types of traces. These traces can record events that take place at several points in a network as data travels from a virtual machine, through VTAM and NCP, to an SNA device. Among those items you can trace in an SNA environment are:

- Buffer contents
- Input/output events
- Line activity
- DFSMS/VM® buffer use
- Transmission group activity
- Internal VSCS and VTAM events.

Detailed information is available in the *VTAM Diagnosis Guide* and the *VTAM Diagnosis Reference*.

Symptom Records

A symptom record is a collection of data conveying basic information about the z/VM software problem. Use the Dump Viewing Facility and the VM Dump Tool to display this data. See the *z/VM: Dump Viewing Facility* and the *z/VM: Dump Viewing Facility* books.

Determining the Cause

After you identify the type of problem, you must determine its cause. There are recommended procedures to follow. These procedures are helpful, but do not identify the cause of the problem in every case. Be resourceful. Use whatever data you have available. If you do not find the cause of the problem after following the recommended debugging procedures, you may need to perform desk-checking.

The section “How to Use z/VM Facilities to Debug” on page 13 describes procedures to follow in determining the cause of various problems that can occur in CP or in the virtual machine. See “Commands That Monitor Events” on page 29 for information on using the CP TRACE command to debug a problem program.

Table 1 on page 3 summarizes the types of problems you may encounter in z/VM.

Data You Need Before Calling IBM for Assistance

If you need to call IBM software support for assistance, it is very important for you to have the following information:

- A problem inquiry data sheet
- A list of all applied maintenance for the module(s) involved
- The operator’s console log
- Verification that all known errors against the Program Update Tape (PUT) have been applied
- The load map for the failing system.

Problem Inquiry Data Sheet

The problem inquiry data sheet (see Figure 1 on page 13) identifies information that should be available to ensure that you get the correct solution from IBM. It might be a good idea to make copies of the sheet, to have blank sheets available in case you have to call IBM.

System Information: When completing the problem inquiry data sheet, you should use the QUERY CPLEVEL command to help you determine these facts about your system:

- The version, release and modification level
- The service level.

For example, if you were on a z/VM system and you entered

```
query cplevel
```

you would get:

```
z/VM VERSION v RELEASE r.m, SERVICE LEVEL yynn (64-bit)
GENERATED AT mm/dd/yy hh:mm:ss timezone
IPL AT mm/dd/yy hh:mm:ss timezone
```

v identifies the software version level.

r.m
identifies the software release level and the release modification level.

SERVICE LEVEL *yynn*
identifies the software service level number. The number indicates the most recent RSU service tape that has been applied. *yy* is the last 2 digits of the year and *nn* is the sequential number of the RSU tape for that year. It cannot indicate which individual updates have been incorporated into CP. The system programmer can find out what individual updates have been incorporated by using the VMSES/E tool. For more information, see the *z/VM: Service Guide*.

GENERATED AT *mm/dd/yy hh:mm:ss timezone*
GENERATED AT *mm/dd/yyyy hh:mm:ss timezone*
GENERATED AT *yyyy-mm-dd hh:mm:ss timezone*
indicates the date and time (translated to the current active time zone) that the CP system software was written to DASD. One of the above responses is generated depending on the date format specified on the user's default date format.

IPL *mm/dd/yy hh:mm:ss timezone*
IPL *mm/dd/yyyy hh:mm:ss timezone*
IPL *yyyy-mm-dd hh:mm:ss timezone*
indicates the date and time the CP system software was last started. One of the above responses is generated depending on the date format specified on the user's default date format.

Record this information on the problem inquiry data sheet.

CPU Information: The QUERY CPUID command should be used to help you to determine what to enter for the CPU serial on the problem inquiry data sheet.

If you entered

```
query cpuid
```

you get:

```
CPUID = FF12069A20848000
```

This is the 16-digit processor identification associated with the real machine. Ignore the FF, which refers to a second level system. The 10 digits that follow the FF are the CPU serial:

- The first six digits are the processor identification number
- The next four digits are the processor model number.

Ignore the last four digits of this 16-digit field.

Note: You can also obtain the system release level, service level, and CPU serial number through the Dump Viewing Facility or VM Dump Tool if a dump was created for the problem. See the description of the SYMPTOM subcommand in the *z/VM: Dump Viewing Facility* and *z/VM: VM Dump Tool* books for more information.

Problem Inquiry Data Sheet Fields: The problem inquiry data sheet consists of the following fields:

Customer
Enter the name of your business.

Date
Enter today's date.

Problem #
Enter the problem number that IBM assigns to you when you call.

Access Code
Enter the customer number that the IBM marketing representative gives to you.

CPU Serial
Enter the 10-digit number from using the QUERY CPUID command, as described above.

Severity
Enter 1, 2, 3, or 4. The severity codes mean:

1 You are unable to use the program, resulting in a critical impact on your operations.

2 You are able to use the program, but you are severely restricted.

3 You are able to use the program with limited functions that are not critical to overall operations.

4 You have found a way to circumvent the problem.

Operating System, Service Level, and Release Level
Enter the system information exactly as displayed in the first line of output from the QUERY CPLEVEL command.

Failing Component
Enter the name of the component that you suspect is causing the problem (for example, CP, CMS, TSAF). Include service level, release level, and other information as appropriate.

Problem/Inquiry Description
Enter the reason for calling IBM software support.

Keywords
Indicate words that best describe the problem, using the provided checklist.

Documentation Available
Indicate the available documentation, using the provided checklist.

Problem Tracking
Enter a log of your activity on the problem, including dates, names, and activity.

Resolution APAR #
Enter the APAR number assigned to the problem (if defect-related).

RSU Tape PTF #
Enter the RSU tape number on which the PTF for the resolution APAR resides.

Other
Enter any other information pertinent to this problem.

Sheet 1 of ___

Customer:	Date:	Problem #:
Access Code:	CPU Serial:	Severity:

Output from QUERY CPLEVEL command :

Failing Component:

Problem/Inquiry Description:

Keywords:

Abend: _____ Module: _____ Wait State Code: _____

Label: _____ Label: _____ Label: _____
Loc: _____ Loc: _____ Loc: _____

Loop Addresses:
_____ Incorrect Output (INCORROUT): _____
_____ Message: _____
_____ Performance: _____

Documentation Available:

Storage Dump	___	User's Routine	___	Console Log	___
Program Listing	___	System Log	___	RSU Level	___
Storage Map	___	Diagnostic Output	___	Service Level	___
Test Data	___	TP CONFIG List(s)	___	VMLOAD List	___

Problem Tracking:

Date	Name	Activity

Resolution RSU Tape
APAR # _____ PTF # _____ Other _____

Figure 1. Problem Inquiry Data Sheet. Use this sheet to collect pertinent data before calling IBM.

How to Use z/VM Facilities to Debug

After you have identified the problem and the area where it occurred, you can gather the information needed to determine the cause of the problem. The type of information you want to look at varies with the type of problem. The tools used to gather the information vary depending upon the area in which the problem occurs. For example, if the problem is a loop condition, you will want to examine the PSW.

For a CP loop, an authorized user's console must be used to display the PSW, but for a virtual machine loop you can display the PSW by using the CP DISPLAY command.

If a procedure tells you to dump storage using the CP DUMP command, you should refer to Chapter 4, "Creating a Dump," on page 49.

Abends

The following types of abnormal terminations (abends) can occur in z/VM:

- CP
- CF service machine
- CMS
- SFS or CRR Server
- GCS
- TSAF
- AVS
- Virtual machine.

Whenever a program abnormally terminates, a message is issued. This message provides information that can help you correct the problem. The following descriptions provide guidelines for debugging each type of abend.

CP Abend

z/VM abnormally terminates when system integrity may be jeopardized. When this happens, a dump is taken. Internal checks on control block fields often determine whether CP issues an abend.

An abend dump includes two primary sources of diagnostic information:

- An abend code
- Symptom record information.

The abend code tells what module has issued the dump and what actions CP is taking or has taken. The format of a CP abend code is:

mmm###

where:

mmm identifies which module issued the abend. The complete module name is prefaced by HCP (for example, HCPmmm).

is the code number.

For example, abend `FRE001` means that CP module HCPFRE issued the abend and 001 is the code number.

When the system terminates abnormally, you receive an error message. For an explanation of error messages and abend codes, see the *z/VM: CP Messages and Codes* book. The explanation for the abend code gives you a start in performing diagnosis.

z/VM issues two types of abends—hard and soft.

Hard Abend

z/VM issues a hard abend when it cannot isolate the error to a single virtual machine. CP dumps all CP and free storage to a dump device. You can set the dump device either at initialization or with the CP SET DUMP command. See the *z/VM: CP Commands and Utilities Reference* for a description of the SET DUMP command.

Soft Abend

z/VM issues a soft abend when CP can isolate the error to a virtual machine or when system integrity is not jeopardized by the error. A soft abend dump results, giving only selected CP pages.

Reasons for the CP Abend

CP will stop and take an abnormal end dump under three conditions:

1. Program check in CP

   Examine the program old PSW and the program interrupt code fields in the prefix page (or page 0) to determine the failing module.

2. Module issuing the HCPABEND macro

   Examine the SVC old PSW and abend code fields in the prefix page (PFXABEND) of the dump to determine the module that issued the abend (SVC 4 for a soft abend) and the reason it was issued.

3. Operator forcing a CP system restart on the processor console

   Examine the restart old PSW field in the prefix page to find the location of the instruction that was processing when the operator forced a CP system restart. The operator forces a CP system restart when CP is in a disabled wait state or loop. Refer to your processor manual for the appropriate method to force a CP system restart.

Use the dump to determine why CP terminated and then determine how to correct the condition.

The DUMPLOAD utility lets you load the dump file from a spooled reader file. The VMDUMPTL command can be used to display information from a CP dump. See the *z/VM: CP Commands and Utilities Reference* for more information on the DUMPLOAD utility. See the *z/VM: VM Dump Tool* for information on the VMDUMPTL command and its subcommands and macros.

CF Service Machine Abend

For information on CF service machine abends, see Chapter 6, “Debugging CF Service Machine Problems,” on page 67.

CMS Abend

For information on CMS abends, see Chapter 7, “Debugging CMS,” on page 69.

SFS or CRR Server Abend

For information on SFS or CRR recovery server abends, see Chapter 9, “Debugging the SFS Server or CRR Recovery Server,” on page 95.

GCS Abend

For information on GCS abends, see Chapter 10, “Debugging GCS,” on page 103.

TSAF Abend

For information on TSAF abends, see Chapter 11, “Debugging TSAF,” on page 181.

AVS Abend

For information on AVS abends, see Chapter 12, “Debugging AVS,” on page 189.

Virtual Machine Abend (Other than CMS)

The abnormal termination of an operating system (such as z/OS or VSE) running under CP appears the same as termination of the operating system on a real machine. Refer to publications for that operating system for debugging information. However, all of the CP debugging facilities may be used to help you gather the information you need.

The CP VMDUMP command dumps virtual storage to a specified virtual machine's reader spool file. You can use the DUMPLOAD utility described in the *z/VM: CP Commands and Utilities Reference* to process the file created by the VMDUMP command.

If you choose to run a stand-alone dump program to dump the storage in your virtual machine, be sure to specify the NOCLEAR option (which is the default) when you enter the CP IPL command. Although CP's IPL simulator program is loaded into a 4 KB page of the virtual machine's virtual storage, CP restores the page to its pre-IPL contents.

If the problem can be reproduced, it may be helpful to trace the processing using the CP TRACE commands. Also, you can display and alter registers, control words (such as the PSW), and data areas. The CP TRACE commands can be very helpful in debugging because you can gather information at various stages in processing. A dump is static and represents the system at only one particular time. Debugging on a virtual machine can often be more flexible than debugging on a real machine.

z/VM may stop a virtual machine if an irrecoverable machine check occurs in that virtual machine. Hardware errors usually cause this type of virtual machine termination. Such errors place the virtual machine into console function mode where it can be made to continue processing on the main processor if you enter the CP BEGIN command. In some cases a check-stopped virtual machine may be indicative of a more pervasive error. A damaged page in an NSS might affect many logged on users. Each user trying to use the NSS could be check-stopped in turn. In another example, a product, such as VTAM running in a check-stopped Service Virtual Machine (SVM) could cause an outage for each and all of its users.

Unexpected Results

The type of errors classified as unexpected results can range from operating systems improperly functioning under CP to printed output in the wrong format.

If an operating system runs properly on a real machine but does not run properly with CP, a problem exists. Also, if a program runs correctly under control of a particular operating system on a real machine but does not run correctly under the same operating system with CP, a problem exists.

First, there are conditions (such as time-dependent programs) that CP does not support. Be sure that one of these conditions is not causing the unexpected results in CP. See the *z/VM: CP Planning and Administration* book for a list of the restrictions.

Next, be sure that the program and operating system running on the virtual machine are the same as those that ran on the real machine. Check for the same:

- Job stream
- Copy of the operating system (and program)
- System libraries.

If you still cannot find the problem, look for an I/O problem. Try to reproduce the problem while tracing all virtual I/O instructions and interrupts with the CP TRACE command. Compare the trace entries. A discrepancy may indicate that one of the CP restrictions was violated, or that an error occurred in CP. Remember, however, that some virtual machines may produce test subchannel (TSCH) or test I/O (TIO) loops while waiting for I/O to complete. This is often an usual occurrence and does not necessarily signify an endless loop.

If unexpected results occur (such as TEXT records interspersed in printed output), you may wish to examine the contents of the system or user files. Non-CMS users may run any of the utilities included in the operating system they are using to examine and rearrange files. See the utilities publication for the operating system running in the virtual machine for information on how to use the utilities.

CMS users should use the DASD Dump/Restore (DDR) utility to print or move the data stored on direct access devices. See the *z/VM: CP Commands and Utilities Reference* for more information on the DDR utility.

Loops

A loop occurs primarily when an instruction sets or branches on a condition incorrectly. You can usually recognize the existence of a loop when productive processing ceases and the program continually repeats the same series of PSW instruction addresses. If I/O operations are involved and the loop is very large, it may be extremely difficult to define, and may even include nested loops. The problem in loop analysis is finding either the instruction that should open the loop or the instruction that passes control to the set of looping instructions. To help you find the problem in a loop, you may want to spool your console to record the instructions or trace the instructions to the printer.

CP Disabled Loop

The processor operator should perform the following sequence when gathering information to find the cause of a disabled loop:

1. Trace the instructions currently running in the processor.
2. Force a CP system restart to cause an abend dump to be taken.
3. Save the information collected for the system programmer or system support personnel.

After the processor operator has collected the information, the system programmer or system support personnel should examine it:

1. Use the instructions traced by the operator and the load map to determine the modules that may be involved in the loop.

2. If the cause of the loop is not apparent, examine the CP internal trace table in the dump to determine the modules that may be involved in the loop.
3. Other information in the dump can be used to determine the condition that caused the loop, such as:
   - PSW
   - General purpose registers
   - Control registers
   - Access registers
   - Prefix page(s) of each CPU.

Virtual Machine Disabled Loop

When a disabled loop in a virtual machine exists, the virtual machine operator cannot communicate with the virtual machine's operating system. This means that signalling attention does not cause an interrupt.

The virtual machine operator should perform the following sequence when trying to find the cause of a disabled loop:

1. Enter the CP console function mode.
2. Use the CP TRACE command to trace the entire loop.
3. USE the CP DISPLAY command to display general purpose and control registers as appropriate depending on when and how they are used.
4. Use the CP DUMP or CP VMDUMP command to dump your virtual storage. If VMDUMP was used, use the DUMPLOAD utility to put the dump onto a disk. For a dump of a ESA/390 Architecture guest, you can use the Dump Viewing Facility or the VM Dump Tool to analyze the dump. For a dump of z/Architecture guest, you must use the VM Dump Tool. For details, see the *z/VM: Dump Viewing Facility* or *z/VM: VM Dump Tool* book.
5. Examine the source code, if available.

Use the information just gathered, along with listings, to try to find the entry into the loop.

If the operating system in the virtual machine itself manages virtual storage, it is usually better to use that operating system's dump program. CP does not retrieve pages that exist only on the virtual machine's paging device.

Virtual Machine Enabled Loop

The virtual machine operator should perform the following sequence when trying to find the cause of an enabled loop:

1. Use the CP TRACE command to trace the entire loop. Display the PSW and the general purpose and control registers.
2. Use the CP DUMP or CP VMDUMP command to dump your virtual storage. If VMDUMP was used, use the DUMPLOAD utility to put the dump onto a disk. For a dump of a ESA/390 Architecture guest, you can use the Dump Viewing Facility or the VM Dump Tool to analyze the dump. For a dump of z/Architecture guest, you must use the VM Dump Tool. For details, see the *z/VM: Dump Viewing Facility* or *z/VM: VM Dump Tool* book.
3. Consult the source code to search for the faulty instructions, examining previously ran modules if necessary. Begin by scanning for instructions that set the condition code or branch on it.

4. If the manner of loop entry is still undetermined, assume that a wild branch has occurred and begin a search for its origin.

Wait States

No processing occurs in the virtual machine when it is in a wait state. When the wait state is an enabled one, an I/O interrupt causes processing to resume. Likewise, when CP is in a wait state, its processing ceases.

To help identify a wait state in your virtual machine, you can periodically enter the command:

```
#cp indicate user
```

to display the resources used by the program. Compare the following resources:

- IO, which is the total number of nonspooled I/O requests issued
- READS, which is the total number of page reads that have occurred
- WRITES, which is the total number of pages written.

When these resources don't change, the wait state probably exists.

CP Disabled Wait

CP enters a disabled wait state when system operation ends because of an error or when system shutdown is complete. When CP or one of its service programs enters a disabled wait state, it loads a wait state code into the program status word (PSW). This PSW appears on your console at the end of the wait state message you receive. For a description of the disabled wait state code and suggested actions to take, refer to the message that has the same number as the wait state code. For example: if the wait state code was 1010, you would look up message HCP1010 in the *z/VM: CP Messages and Codes* book.

A disabled wait state usually results from a hardware malfunction. Most disabled wait states occur during the initial program load (IPL) process. Many can be attributed to normally correctable hardware errors that may cause a wait state because the operating system error recovery procedures are not yet accessible. Other frequent disabled wait states during IPL involve the system resident device (SYSRES), which may have been formatted improperly, defined with the wrong device type, or may have experienced an I/O error.

Disabled wait code 1010 is often found when installing a z/VM system for the first time. This code indicates that no console was available; typical reasons are:

- No definition for a console on the OPERATOR_CONSOLES statement in the system configuration file or the console was defined incorrectly
- If running in virtual mode, the CP TERMINAL CONMODE 3270 command was not entered or a CP DEFINE CONSOLE command was entered incorrectly.

Codes 961, 964, and 9025 are common and can occur after the system is shut down.

A severe machine check during post-IPL processing can also cause a CP disabled wait state.

CP Enabled Wait

If you determine that CP is in an enabled wait state, but that no I/O interrupts are occurring, either there may be an error in CP or CP may be failing to get an interrupt from a hardware device. Force a CP system restart at the operator's console to cause an abend dump to be taken. Use the abend dump to determine the cause of the enabled (and noninterrupted) wait state. After the dump is taken, IPL the system.

Using the dump, examine the:

- Virtual machine definition blocks (VMDSCAN)
- Real device block (RDEVBK).

See "Reading CP Abend Dumps" on page 55 for specific information on how to analyze a CP dump.

Virtual Machine Disabled Wait

CP does not allow the virtual machine to enter a disabled wait state or certain interrupt loops. Instead, CP notifies the virtual machine operator of the condition with one of the following messages:

```
HCPGIR450W   CP entered; disabled wait PSW psw
HCPVIX452I   CP entered; external interrupt loop
HCPGIR453W   CP entered; program interrupt loop
```

and enters the console function mode.

An explanatory message from the operating system running in your virtual machine may precede the HCPGIR450W message. If you did not receive an explanation, examine the PSW portion of the message. To interpret the wait state code in the PSW, refer to the section on wait states of the corresponding manual for the system you were running in your virtual machine. Take the specified corrective action, then re-IPL the virtual system.

An Example of a Virtual Machine Disabled Wait

You were running CMS and received the message:

```
HCPGIR450W   CP entered; disabled wait PSW 000A0000 00000070
```

This means that CMS received a virtual machine check. Re-IPL CMS and try again.

For message HCPVIX452I, determine why the external interrupt new PSW is enabled for an interrupt condition that does not clear upon acceptance (that is, the timer is not expected to contain a negative value).

To determine the reason for message HCPGIR453W, examine the program check information in page zero of your virtual storage. If this error occurred immediately after the IPL command, the problem may be that you are trying to run a System/390® guest in an XC virtual machine, or the reverse. To correct this error, enter:

1. The CP QUERY SET command to find out the current MACHINE setting.
2. The CP SET MACHINE command to select the proper virtual machine.

If the virtual machine was running disconnected when the loop occurred, the system logs it off. If this happens, you may need to reproduce the interrupt loop with the virtual machine running connected to a console. To continue, IPL the virtual system again.

To examine the contents of storage locations, registers, and control words on a terminal, use the CP DISPLAY command. Some of the data you can see includes:

- The program status words
- The general-purpose registers
- The control registers
- The storage contents of your virtual machine.

Then use the CP DUMP or CP VMDUMP command to dump your virtual storage. If VMDUMP was used, use the DUMPLOAD utility to put the dump onto a disk. For a dump of a ESA/390 Architecture guest, you can use the Dump Viewing Facility or the VM Dump Tool to analyze the dump. For a dump of z/Architecture guest, you must use the VM Dump Tool. For details, see the *z/VM: Dump Viewing Facility* or *z/VM: VM Dump Tool* book.

If you cannot find the cause of the wait or loop from the information just gathered, try to reproduce the problem, this time tracing the processing with the CP TRACE command.

If CMS is running in the virtual machine, you may also use the CMS debugging facilities to display information or trace the processing. See “Using CMS to Debug” on page 77 for more information.

Virtual Machine Enabled Wait

If the virtual machine is in an enabled wait state, try to find out why no I/O or external interrupts have occurred to allow processing to resume.

CP treats one case of an enabled wait in a virtual machine the same as a disabled wait. If the virtual machine does not have the “real timer” option, CP issues the message:

```
HCPGIR450W   CP entered; disabled wait PSW psw
```

Because the virtual timer is not decreased while the virtual machine is in a wait state, it cannot cause the external interrupt. The “real timer” runs in both the problem state and wait state and can cause an external interrupt that allows processing to resume. The clock comparator can also cause an external interrupt.

Hang Conditions

A hang condition occurs when either CP cannot continue processing or a virtual machine cannot be dispatched. As a result, z/VM halts processing.

When gathering data about hang conditions, keep in mind that a delay may occur between the time the error-causing request is issued and the time the system hangs. The module running when the hang occurs may not be the module responsible for the hang. As a result, some tools may provide no useful diagnostic data. For example, CP continuously creates trace entries in a trace table for each active processor in your configuration. Later trace entries may be written over the trace entry describing the event that caused the hang.

There are two types of hangs:

- System
- User.

System Hangs

System hangs occur when z/VM cannot perform any tasks to completion.

The best way to handle a system hang is for the hung system's operator to restart z/VM from the operator's console. At that point, CP issues an SVC002 abend dump and attempts a restart.

Diagnosing the cause of a system hang can be difficult. The following actions are starting points:

- Locate the active virtual machine descriptor block (VMDBK) to determine which user was running at the time of the dump. By looking at the scheduling controls (VMDSLIST and VMDSTATE) in that VMDBK, you can determine if this was the active VMDBK and what the user was doing.

  You can use the VMDUMPTL command of the VM Dump Tool for this. See the *z/VM: VM Dump Tool* for more information about the VMDUMPTL command.
- Check the restart old PSW. It points to the last instruction before the restart.
- Examine any trace entries available.

User Hangs

A user hang occurs when a virtual machine is no longer dispatched by CP. You need to determine if the hang was caused by z/VM or the operating system you are running in the virtual machine. The first step is to look at the operating system running in the virtual machine to determine if it is hung.

One way of determining that the virtual machine is hung is to attempt a #CP command. (For more information on issuing CP commands with #CP, see the *z/VM: CP Commands and Utilities Reference*.) For instance, entering the command:

```
#cp indicate user
```

causes one of two things to appear on your screen if you are in line mode:

1. Information about your virtual machine, if it is not hung
2. Nothing, if your virtual machine is hung.

If your virtual machine appears to be hung and it is not, you can enter the command:

```
#cp indicate queues
```

If the user is in the eligible list, then over-committing storage by entering the SET SRM STORBUF command can move the user off the eligible list and onto the dispatch list. See the *z/VM: Performance* book, specifically the section on tuning the storage subsystem for more information. As with a system hang, the best source of information is the VMDBK. From an authorized user, locate the hung user's VMDBK. Check the scheduling and dispatching controls (VMDSLIST and VMDSTATE) in the hung user's VMDBK to determine what state the user was in when the hang condition occurred. If you cannot free the user based on the cause of the hang condition, you may need to force the user off and log the user on again. As a last resort, you may need to restart z/VM from the operator's console. This will create an SVC002 abend dump that can be used to do more diagnosis.

Use of z/VM Debugging Commands

There are many commands that are useful for interactively debugging a problem. The chapters that follow contain many examples of commands that can be used with the different components of z/VM. However, the commands that you use are not limited to the examples that are given. Any commands or locally produced routines can be used for debugging a problem.

Chapter 2. Debugging Interactively

CP provides interactive commands that control the system and enable the user to control his virtual machine and associated control program facilities. The virtual machine operator using these commands can gather much the same information about his virtual machine as the operator of a real machine gathers using facilities on the processor console.

Several of these commands (for example, CP DISPLAY or CP STORE) examine or alter virtual storage locations. When CP is in complete control of virtual storage (for example, as in the case of CMS and GCS) these commands run as expected. However, when the operating system in the virtual machine itself manipulates virtual storage (for example, as in the case of MVS or VSE), you should be very cautious if you use these CP commands.

This chapter presents an overview of the z/VM commands used for debugging. Instructions for using the commands discussed in this chapter are in the following books:

- *z/VM: CP Commands and Utilities Reference*
- *z/VM: Dump Viewing Facility*.

You can use the following categories of commands to help diagnose problems interactively:

- Commands that display and dump machine data
- Commands that set and query system features, conditions, and events
- Commands that monitor events
- Commands that alter the contents of storage
- Commands to collect and alter system information.

Commands That Display and Dump Machine Data

The CP DISPLAY command allows a user to display data from several real and virtual machine components at a terminal. The CP DUMP command allows a user to print data from several real and virtual machine components at a printer. The data that can be displayed or printed is controlled by the privilege class of the user. See the *z/VM: CP Commands and Utilities Reference* for more information on these commands.

Use the CP DISPLAY command to display the following kinds of control information at your terminal or the CP DUMP command to print the following kinds of control information on a printer.

- The contents of first-, second-, and third-level storage
- The contents of storage in address spaces of XC virtual machines
- Storage keys
- Prefix register
- General purpose registers (GPRs)
- Floating-point registers
- Control registers
- Access registers
- Crypto domain index registers

- PSW
- The subchannel information blocks (SCHIBs)
- Linkage stacks
- Virtual machine host access list.

Terminal Output

You can use the DISPLAY command to examine the general purpose registers, floating-point registers, control registers, access registers, and crypto domain index register. For example, the commands:

```
display gg
display g
display g1
display g2-5
display y
display x7
display ar
display cdx
```

result in displays of all the GPRs (display gg or display g), GPR1, a range of GPRs 2 through 5, all the floating-point registers, control register 7, all access registers, and the crypto domain index register, respectively.

The DISPLAY command also displays the PSW and SCHIB:

```
display pswg
display psw
display schib
```

Class G users can display virtual machine storage information. Class C or E users can display first level-storage information by using the DISPLAY H command. The examples that follow are examples of virtual machine storage. First-level storage output is similar except that the displayed line begins with H instead or R. The storage information is displayed at your terminal in either of the following formats:

- Four-byte groups, aligned on fullword boundaries, hexadecimal format, with four fullwords per line. For example, if you enter the DISPLAY command as:

  ```
  display 1026-102c
  ```

  you receive the response:

  ```
  R00001024 xxxxxxxx xxxxxxxx xxxxxxxx          F6
  ```

- 16-byte groups, aligned on 16-byte boundaries, hexadecimal format, with four fullwords and EBCDIC translation per line. For example, if you enter the DISPLAY command as:

  ```
  display t1026-102c
  ```

  The response is:

  ```
                                                  (EBCDIC trans.)
  R00001020 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx F6 *.............*
  ```

You can also specify the area of storage to be displayed by entering a hexadecimal byte count such as:

```
display 1024.12
```

The response displays 20 bytes as follows:

```
R00001024 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx F6
R00001034 xxxxxxxx
```

In addition, the storage key is displayed on the first line, as well as at every page boundary.

The previous responses illustrate the byte alignment that takes place in each of the two display formats.

If the first location to be displayed is not on the appropriate 4- or 16-byte boundary, it is rounded down to the next lower boundary that applies.

If the last location to be displayed does not fall at the end of the appropriate 4- or 16-byte group, it is rounded up to the end of that group.

If you enter:

```
display k1024-3800
```

the storage keys that are assigned to each 4K segment of the specified storage area are displayed. For example, the response might be:

```
R00001000 TO 0037FF  KEY=F6
R00003800 TO 003800  KEY=E0
```

Contiguous 4K segments with identical storage keys are combined.

To display all storage keys, enter:

```
display k0-end
```

You can display any of the control registers. For example, enter:

```
display x1 4 a
```

and receive the response:

```
ECR  1 = xxxxxxxx
ECR  4 = xxxxxxxx
ECR 10 = xxxxxxxx
```

Printer Output

With the DUMP command you can dump the contents of all available registers, the PSW, the virtual machine's host access list, and the storage keys, along with any specified area of virtual storage, to the virtual machine's printer.

To print only the registers, the PSW, and the storage keys, you need only enter:

```
dump 0
```

To also print an area of virtual storage, you can specify the beginning and ending hexadecimal locations:

```
dump 1064-10ff
```

You can also specify in hexadecimal the beginning location and the number of bytes to be dumped:

```
dump 1064.9b
```

If you are printing a series of dumps, you can identify each one by including its identification on the DUMP command line, following an asterisk:

```
dump 1000-2000 * dump no. 1
```

When you direct dump output to a printer, the dump output is mixed in with any printed program output. If you want dump output separated from other printed output, use the CP DEFINE command to define a second printer. Dump output is always sent to the virtual printer having the lowest address, so you must define the dump printer at a address below the one used for program output. If the printer is defined in the z/VM director as address 00E and you enter:

```
define printer 006
```

The dump output will go to the printer at address 006 and any other printed output will go to the printer at address 00E.

To print the dump data on the real printer you must first close the virtual printer. Enter:

```
close 006
```

This closes the dump data spool file and releases it for processing on a real printer.

You can use the CP VMDUMP command to dump the storage of your virtual machine. Then use the DUMPLOAD utility to put the dump onto a disk. For a dump of a ESA/390 Architecture guest, you can use the Dump Viewing Facility or the VM Dump Tool to analyze the dump. For a dump of z/Architecture guest, you must use the VM Dump Tool. For details, see the *z/VM: Dump Viewing Facility* or *z/VM: VM Dump Tool* book.

When you enter at the terminal:

```
vmdump 150-200
```

or

```
vmdump 400:500
```

CP dumps the contents of virtual machine storage at the hexadecimal addresses between X'150' and X'200' or between X'400' and X'500', respectively.

If you enter:

```
vmdump 150.50
```

CP dumps the contents of virtual storage starting at X'150' for a total of X'50' bytes.

If you enter:

```
vmdump 150.all
```

CP dumps the contents of virtual storage from location X'150' to the end of the virtual machine address space, including guest storage and all the DCSSs above guest storage.

Commands That Set and Query System Features, Conditions, and Events

The SYSTEM and SET commands set system-controlled functions and events; the QUERY command lets you determine the status of those settings.

The SYSTEM command is a privilege class G command that simulates the RESET and RESTART functions on a real computer console. You can also use it to clear

storage and store status in a virtual machine. The SYSTEM command is described in the *z/VM: CP Commands and Utilities Reference*.

Some operands of the SET command useful for debugging are MSG, SMSG, WNG, EMSG, and IMSG. The messages resulting from these settings may be useful to you while you are debugging.

The SET MSG function determines whether you receive messages sent by other users by way of the MSG command.

The SET SMSG command turns on or off a virtual machine's special message flag. If the virtual machine has issued DIAGNOSE code X'68' (AUTHORIZE), this flag determines whether the virtual machine accepts or rejects messages sent by way of the SMSG command — when the flag is on, messages are accepted.

The SET WNG function determines whether you receive warning messages.

The SET EMSG command controls error message handling. Messages can be displayed in several ways depending upon how this command is entered. If SET EMSG ON is specified, both the message identifier and text are displayed. If SET EMSG TEXT is specified, only the message text is displayed. If SET EMSG CODE is specified, only the identifier is displayed. If SET EMSG IUCV is specified, both the error code and text are passed to the virtual machine through IUVC if a connection to the message system service exists. If no IUCV connection exists, the message is handled as if SET EMSG ON had been entered. You can also specify SET EMSG OFF so that no error messages are displayed. When you log on, EMSG is set to ON. Because it displays the complete message, this setting is useful when you are debugging. The information contained in the message identifier is especially helpful. It contains the name of the component and module that issued the message as well as a message number which makes it easier to locate in the *z/VM: CP Messages and Codes* book.

The SET IMSG command controls whether certain informational responses issued by some CP commands are displayed at the terminal. Also, the SET IMSG command determines whether you receive messages from CP when other users spool reader, printer, or punch files to your virtual machine.

When you are debugging, it may be useful to have all messages displayed at your terminal.

The SET RUN command controls whether the virtual machine stops when the attention key is pressed.

The QUERY command displays the status of features and conditions set by the SET command for your virtual machine. When you log on, the MSG, EMSG, IMSG, and WNG operands of the SET command are set ON, and the SMSG and RUN operands are set OFF. To verify these settings, use the QUERY SET command.

Commands That Monitor Events

The TRACE command monitors events that occur in your virtual machine. Some of the events that you can trace include:

- Instruction processing
- Storage alteration
- Register alteration

- I/O activity.

The TRACE command has many options. The primary operands allow you to selectively choose the events to monitor. Each of the primary operands used with the TRACE command establishes *trace traps*. A trace trap is a programming function that captures information about an event in your virtual machine. For example, to trace all events, enter:

```
trace all
```

To trace supervisor calls, program interrupts, and changes to the access registers, enter:

```
trace svc
trace prog
trace ar
```

Continuing with this example, if, after specifying multiple activities to be traced, you decide to stop tracing one or more of them, enter:

```
trace delete trap1
trace delete trap2
```

where *trap1* and *trap2* are the identifiers for the program interrupt and access register trace traps. Tracing is now confined to SVCs only.

You can also specify multiple trace events on a single command by using the TRACE GOTO command to specify the name of a trace set that contains a list of trace commands to be run. To define the named trace set, enter:

```
trace goto name
trace svc
trace prog
trace ar
```

To activate the named trace set, enter:

```
trace call name
```

To end the named trace set, enter:

```
trace end
```

or

```
trace return
```

Controlling the Trace Information

There are several common options for controlling the amount of information you receive when you are using the TRACE command and how the information is received.

For example, whenever you are recording trace output to display at your terminal, the virtual machine stops running and enters the CP console read environment after each output line. If you do not want program processing to halt every time a trace output message is issued to the terminal, you can use the RUN option:

```
trace svc run
```

In the above example, the RUN option is used with a SVC trace. Entered in this way, the command lets you watch supervisor call activity in your program without halting processing every time a call occurs.

If you do not require your trace output immediately, you can direct it to the printer, so that your terminal does not receive any information at all. Also, tracing to the terminal takes you out of fullscreen mode. You may want to direct your trace output to the printer to preserve the fullscreen environment if you are tracing a fullscreen application (for example, XEDIT):

```
trace inst printer
```

When you direct trace output to a printer, the trace output is mixed in with any printed program output. If you want trace output separated from other printed output, use the CP DEFINE command to define a second printer. Trace output is always sent to the virtual printer having the lowest address, so you must define the trace printer at a address below the one used for program output. If the printer is defined in the z/VM director as address 00E and you enter:

```
define printer 006
```

The trace output will go to the printer at address 006 and any other printed output will go to the printer at address 00E.

When you finish tracing, use the CP CLOSE command to close the second virtual printer file:

```
close 006
```

If you want trace output at the printer and at the terminal, you can use the BOTH option:

```
trace all both
```

Trace output is always produced **after** the instruction is processed.

Restricting the Trace to an Address Range

The common options more clearly define the trace traps set by the primary operand. The PSWA option lets you restrict instruction tracing to a particular address range. Note that the address range remains in effect until you turn off the trace element set up by the TRACE command.

For example, entering the command:

```
trace instruct pswa 20000
```

causes program processing to halt after the instruction at location X'20000' is processed.

The following command:

```
trace instruct pswa 20000-20400
```

traces all the instructions within the range of X'20000' and X'20400' and produces output for each instruction.

To see what events are currently being traced, enter:

```
query trace
```

For detailed examples of tracing programs in a virtual machine, see the *z/VM: Virtual Machine Operation* book.

Selectivity

You can use many of the TRACE common options to increase selectivity. Using TRACE, it is possible to limit tracing to a specific instruction or set of instructions. For example, to monitor only LR instructions (operation code X'18'), enter:

```
trace instruct data 18
```

When the NORUN option is in effect, program processing halts after each monitored event. When the RUN option is in effect, program processing continues after each event. TRACE also counts occurrences between NORUN and RUN. These options are STEP, STOP, PASS, and SKIP. For example, to halt program processing after 5 instructions in the range X'20000' to X'204FF' have been run, enter:

```
trace instruct pswa 20000.500 step 5
```

Program processing halts and enters the CP command environment.

Although the STEP option lets you step through your program more quickly without giving up all control, every monitored instruction is displayed. If many instructions are processed before the problem occurs, you may need to frequently clear your screen. You can change the frequency with which events are displayed by using the PASS option. Ordinarily, every successful event is displayed. However, using the PASS option makes it possible to specify the number of monitored events you want to skip before displaying one. For example, to skip the display of 100 instructions and display the 101st, enter:

```
trace instruct pass 100
```

Tracing Successful Events

Another method of finding the failing instruction is to use the TRACE COUNT command to count the successful trace events in your virtual machine, and the TRACE TABLE command to display a list of successful branch instructions. If the program is abending with any sort of program exception, load the failing program and enter the CP command:

```
trace prog
```

Follow this with the command:

```
trace instruct range 20000.500
```

(assuming the program is loaded at location X'20000' and is X'500' bytes in length). Then enter the command:

```
trace count
```

Next start the failing program. No trace output is produced while the COUNT option is in effect. When the program interrupt occurs, enter the QUERY TRACE command to display the current count:

```
query trace
```

You can trace the program after using the TRACE PASS option to get close to the problem.

You can also use TRACE COUNT in conjunction with more specific trace elements to produce the desired results. For example, if a problem occurs as a result of processing an SVC 202 and the failing program issues many SVC 202s before failing, trace only SVC 202s (operation code X'0ACA') and use TRACE COUNT to count the occurrences. First, load the failing program and then enter:

```
trace svc aca
trace count
```

and start the program. When the error occurs, enter a QUERY TRACE to check the count.

```
query trace
```

You can trace the program after using the TRACE PASS option to get close to the problem.

For detailed examples, see the *z/VM: Virtual Machine Operation* book.

Tracing Storage Alteration

You can use the TRACE command to trace the alteration of storage in the user's virtual machine. If you specify TRACE STORE, then whenever an instruction places a value into storage, that event is traced. See the usage notes for the TRACE STORE command in the *z/VM: CP Commands and Utilities Reference* for a list of exceptions to the above statement. It is **not** necessary that this value be different from the previous value.

It is also possible to monitor the alteration of storage to a specific value. For example:

```
trace store into 20100 data 112757
```

monitors instructions that cause the storage at location X'20100' to become X'112757'. Note that these instructions are traced even if the value at location X'20100' was already X'112757' before processing any instructions.

The TRACE CMD Option

You can use the CMD option of the TRACE command to run any CP command (except SLEEP) whenever a particular event occurs. For example:

```
trace instruct pswa 20000.500 run
trace store 204f0-204ff pwsa 20000.500 run cmd display 204f0-204ff
```

traces the processing of every instruction in the range X'20000' through X'204FF' and displays the contents of storage at X'204F0' through X'204FF' every time any storage within the range X'204F0' through X'204FF' is altered by an instruction in the range X'20000' through X'204FF'.

Also, you can use the CMD option to allow a program to continue at a specific address whenever a particular event occurs. For example:

```
trace instruct pswa 20000.500 printer
trace branch into 0 run cmd begin 24f28
```

causes program processing to continue at location X'24F28' whenever a branch to location 0 occurs. Processing continues after the instruction is displayed. When program processing resumes at location X'24F28' and a subsequent branch to zero occurs, processing again begins at location X'24F28'. This can result in a loop. You can use the CMD option to prevent this. For example, if LINEDIT is on, and the escape character is set to ″ and the line end character is #, enter:

```
trace instruct 20000.500 printer
trace branch into 0 run cmd trace clear branch"#begin 24f28
```

turns off the branch trace element and causes program processing to continue at location X'24F28' after the instruction is displayed.

The commands associated with each trace element are run whenever the event described by the trace element occurs. The commands are run in the order in which they appear in the set of events.

Note: If you enter a CP command while commands are being processed by TRACE, the output from the commands may be interleaved.

After you have specified the CMD option for a particular trace element, the CMD option remains in effect until the trace element is turned off or until you change it. To change the option, see the *z/VM: Virtual Machine Operation* book.

Stopping the TRACE

When you stop tracing, you must also enter the CLOSE command to release the spooled trace output file for processing:

```
trace end
close vdev
```

For a more complete explanation, see “Controlling the Trace Information” on page 30.

Commands That Alter the Contents of Storage

Altering Contents of Virtual Machine Storage (STORE Guest Command)

Use the CP STORE (Guest Storage) command to alter the contents of specified registers and locations in virtual machine storage. The contents of the following can be altered:

- The contents of second-, and third-level storage
- The contents of storage in address spaces of XC virtual machines
- General purpose registers
- Floating-point registers
- Floating-point control registers
- Control registers
- Access registers
- Crypto domain index registers
- PSW

Virtual storage can be altered in either fullword or byte units.

When using fullword units, the address of the first positions to be stored must have either an L or no prefix. Each fullword operand can be from one to eight hexadecimal digits in length. If less than eight digits are specified, they are right-justified in the fullword unit and padded to the left with zeros. For example, the command:

```
store 1024 46a2
```

or

```
store l1024 46a2
```

results in X'000046A2' being stored in locations X'1024' through X'1027'.

On the other hand, the command:

```
store 1024 46 a2
```

implies storing two fullwords and results in the storing of X'00000046000000A2' in locations X'1024' throughX'102B'.

If the starting location is not a multiple of a fullword, it is automatically rounded down to the next lower fullword boundary.

You can store in byte units by prefixing the start address with an S. The command:

```
store s1026 d1d6c5
```

stores X'D1D6C5' in locations X'1026' through X'1028'. Note that the data storage is byte-aligned. If an odd number of hexadecimal digits is specified, CP does not store the last digit, you receive an error message, and CP ends the function. For example, if you specify:

```
store s1026 d1d6c
```

CP stores d1 at X'1026' and d6 at X'1027'; when CP attempts to store c, it recognizes an incomplete hexadecimal digit, and does not store the last digit.

You can store data into one or multiple consecutive registers.

General and control registers are loaded in fullword units that are right-justified and padded to the left with zeros. For example, entering:

```
store g4 123456
```

loads GPR 4 with X'00123456'. The following command:

```
store g4 12 34 56
```

loads GPRs 4, 5, and 6 with X'00000012', X'00000034', and X'00000056', respectively.

Floating-point registers are loaded in doubleword units. Each doubleword operand can be from 1 to 16 hexadecimal digits in length. If less than 16 digits are specified, they are left-justified in the doubleword unit and padded to the right with zeros. For example:

```
store y2 00123456789
```

loads floating-point register 2 with the value X'0012345678900000'.

Altering Contents of Host Storage (STORE Host Command)

Privilege class C users can use the CP STORE (Host Storage) command to alter the contents of host storage (first-level storage). For example, the STORE (Host Storage) command can be used to alter information in the old and new PSWs. See the *z/VM: CP Commands and Utilities Reference* for details.

Simulating the Hardware Store Status Facility (STORE STATUS)

You can use the STORE STATUS command to simulate the hardware store status facility. Selected virtual machine data is stored in permanently assigned areas in low storage. Enter:

```
store status
```

The data stored by the STORE STATUS command is:

Table 2. Non-z/Architecture mode guest

Address		Length	Content
Dec	Hex		
163	A3	1	Architectural-mode id (X'00')
212	D4	4	Extended save area address. See note below.
216	D8	8	CPU timer
224	E0	8	Clock comparator
256	100	8	Current PSW
264	108	4	Prefix register
288	120	64	Access registers 0 through 15
352	160	32	Floating-point registers 0, 2, 4, 6
384	180	64	General registers 0 through 15
448	1C0	64	Control registers 0 through 15

Note: The extended save area address is used only if it is provided and floating-point extensions are enabled. When the extended save area is available, the virtual machine's floating-point registers 0 through 15 and floating-point control register are stored there.

Table 3. z/Architecture mode guest

Address		Length	Content
Dec	Hex		
163	A3	1	Architectural-mode id (X'01')
4608	1200	128	Floating-point registers 0 through 15
4736	1280	128	General registers 0 through 15
4864	1300	16	Current PSW
4888	1318	4	Prefix register
4892	131C	4	Floating-point control register
4900	1324	4	TOD programmable register
4904	1328	8	CPU timer
4913	1331	7	Clock comparator
4928	1340	64	Access registers 0 through 15
4992	1380	128	Control registers 0 through 15

Note: If the operating system that is running in your virtual machine operates in the basic control mode, these areas of low storage may be used for other purposes. You should not use this facility under these conditions.

For detailed information about these commands, see the *z/VM: CP Commands and Utilities Reference*.

When debugging, you may find it advantageous to alter storage, registers, or the PSW and then restart the program. This is a good procedure for testing a proposed change. Also, you can make a temporary correction and then continue to ensure that the program runs trouble free.

A procedure for using the STORE STATUS command when debugging is as follows:

- Enter the STORE STATUS command before entering a routine you wish to debug.
- When processing stops (because an address stop was reached or because of an error), display the status information that was stored with the STORE STATUS command.
- Enter STORE STATUS again and display the status information that was stored with the STORE STATUS command. You now have the status information before and after the error. This information should help you solve the problem.

STORE STATUS can also be done when taking a stand-alone dump by issuing the command on a CPU where you will IPL the stand-alone dump utility.

Commands to Collect and Analyze System Information

The following commands can be used to collect and analyze system information when debugging:

- MONITOR
- INDICATE
- QUERY SRM
- LOCATE.

The CP MONITOR command provides a data collection tool that samples and records a wide range of data. The CP INDICATE command provides a method to observe the load conditions on the system while it is running. The CP QUERY SRM command provides observation facilities for analyzing internal activity counters and parameters.

See the *z/VM: CP Commands and Utilities Reference* for more information on the MONITOR, INDICATE, and QUERY SRM commands.

See the *z/VM: Performance* book for more information on system tuning and performance.

Use the class C or E CP LOCATE command to find the address of CP control blocks associated with a particular user, a user's virtual device, or a real system device.

What to Do If Your Program Loops

If your program seems to be in a loop, you should first verify that it is looping, and then interrupt its processing and do one of the following:

- Halt it entirely and return to the previous environment
- Restart the program at an address outside of the loop.

An indication of a program loop may be what seems to be an unreasonably long processing time.

If you are in a long loop, you can use the CP TRACE command with the RUN option and look at the addresses run to identify the loop.

In a smaller loop, you can verify a loop by checking the PSW frequently. If the last word repeatedly contains the same series of addresses, it is a good indication that

your program is in a loop. To check the PSW of your virtual machine, you must be in the CP command environment. You can then use DISPLAY PSW to examine the PSW by entering:

```
display psw
```

and then enter the command BEGIN to restart the program:

```
begin
```

If you are checking for a loop, you might enter both commands on the same line using the logical line end. If the line end is set to a pound sign (#), enter:

```
display psw#begin
```

When you have determined that your program is in a loop, you can stop the program by entering the CMS immediate command HX:

```
hx
```

If you want your program to continue at an address past the loop, you can use the CP command BEGIN to specify the address at which you want to continue. For example, enter:

```
begin 20cd0
```

You could also use the CP command STORE to change the instruction address in the PSW before entering the BEGIN command. For example, enter:

```
store psw 0 20cd0#begin
```

Debugging with CP after a Program Check

If a program check occurs while your program is running, your virtual machine may stop with a disabled wait state. To force your virtual machine to stop when a program check occurs, use the TRACE command.

```
trace prog
```

All of your program's registers and storage areas remain exactly as they were when program interruption occurred. The PSW that was in effect when your program was interrupted is in the program old PSW. Enter one of the DISPLAY commands to examine its contents:

```
display psw prog
display pswg prog
```

If, after using CP to examine your registers and storage areas, you can recover from the problem, you must use the STORE command to restore the PSW, specifying the address of the instruction just before the one indicated by the program old PSW. For example, if your program was loaded at X'20000' and the instruction address in your program is X'566' enter:

```
store psw 0 20566
begin
```

In this example, setting the first word of the PSW to 0 turns the wait bit off and clears all other information in the first word, so that processing can resume.

Chapter 3. Using Traces to Debug

When CP tracing is active, system events are recorded as trace entries in trace tables in real storage. The initial number of trace table pages available to a processor is determined by the TRACE portion of the STORAGE statement in the system configuration file. The TRACE portion of the STORAGE statement lets you specify the number of trace table pages for the master processor and a percentage of that number of pages for all alternate processors. The effect of this initial specification can be changed by using the SET TRACEFRAMES command; the values currently in effect can be displayed by using the QUERY TRACEFRAMES command. Trace entries are created for each processor in a configuration as long as tracing is enabled.

Locating the CP Trace Table

CP keeps a detailed record in the CP trace table of every major event that takes place in your real machine. This table is useful, particularly when trying to discover the events that led to an error in CP.

To find the address where trace tables begin, check the value in PFXTTPNT in the prefix page. For additional information on the prefix page, see “HCPPFXPG: The Prefix Page” on page 58.

Control register 12 contains the address at which the next trace entry will be placed. That address, minus X'20' or X'40' (depending on the entry length (see entry formats)) is the address of the last trace entry created.

Note: Ignore bit 31 of control register 12. It is a flag indicating whether tracing is currently active.

Figure 2 illustrates the concepts that each processor in a configuration has its own allotment of trace table pages, that PFXTTPNT points to the beginning of the trace table, and that control register 12 points to the next trace entry.

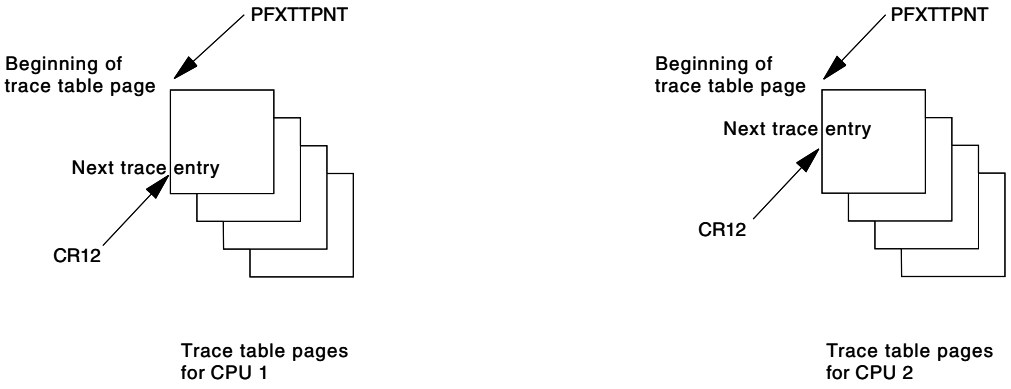


Figure 2. Trace Table Pages for Each Processor

Trace Entries

Trace table entries can be 32 or 64 bytes. An entry’s length and format are defined in its first two bytes:

- the low-order half of the first byte: 7x, where x is the number of register fields minus one
- the high-order bit of its second byte: y0, where y is 1 for 64-byte format and 0 for 32-byte format.

Thus the first two bytes of trace entries are:

- 7400 - 32-byte entries in the format further described below
- 7580 - 64-byte entries (this format is shown in Appendix C)

In addition to these first two bytes, trace table entries contain:

- A time-of-day clock value that indicates when the entry was made
- A constant field (0000)
- A code that defines the event being traced
- A maximum of 40 bytes of information about the specific event traced.

Figure 3 shows the format of a 32–byte trace entry as it would appear in a dump.

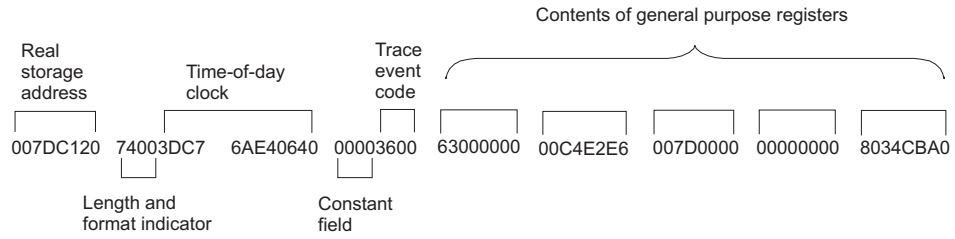


Figure 3. Format of a 32–byte Trace Entry

Each trace entry contains information on a specific system event. Consider the sample trace entry shown in Figure 4:

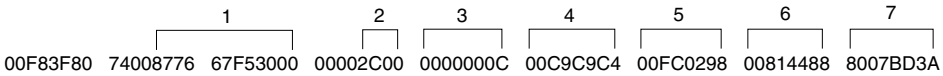


Figure 4. Sample Trace Entry in a CP Abend Dump

In this 32–byte trace entry at address X'00F83F80', the number over the blocks of storage refer to the following items:

1. The time-of-day (TOD), bits 16 through 63, was set to X'877667F53000' when this trace entry was created (at X'02' in the trace entry).
2. The trace event code was X'2C00', a RETURN WITH SAVE AREA (at X'0A' in the trace entry).
3. The value returned in register 15 was X'0000000C' (at X'0C' in the trace entry).
4. The condition code was 0, and the returning module identifier was ‘IID’ (at X'10' in the trace entry).
5. The returned SAVBK address in register 13 was X'00FC0298' (at X'14' in the trace entry).
6. The real address of the calling module from register 14 was X'00814488' (at X'18' in the trace entry).

7. The real exit address of the called module from register 14 was X'8007BD3A' (at X'1C' in the trace entry).

In this example, CP stored the contents of the general purpose registers at X'FC0298' with a return code of 12.

For a complete listing of trace table codes and their field values, see Appendix C, "Trace Table Codes," on page 213.

Limiting the Trace Entries Recorded

Normally, CP tracing is active during system operation. However, new trace entries are added continually to trace tables and eventually are written over older trace entries. This process is called wrapping.

On stressed systems, wrapping may occur in well under one second. As a result, an abend dump that includes the trace table for each processor may convey little or no information about the problem. z/VM overcomes this limitation by allowing class A and C users to do the following:

- Limit tracing to certain user IDs or event codes
- Filter out data for certain user IDs or event codes
- Save entries on tape or in system trace files
- Refine captured information.
- Trace and display real I/O devices
- Trace and display most code paths in CP
- Extract captured trace data, including captured trace table data from trace buffers within a CP dump.

For tracing activities, you mainly use eight CP commands:

- SET CPTRACE
- QUERY CPTRACE
- TRSOURCE
- QUERY TRSOURCE
- TRSAVE
- QUERY TRSAVE
- QUERY TAPES
- QUERY TRFILES.

See *z/VM: CP Commands and Utilities Reference* for the format of, and information about, these commands.

For processing trace data recorded by the TRSOURCE command or for processing CP trace data, you use one CP utility:

- TRACERED.

See *z/VM: CP Commands and Utilities Reference* for detailed information about using the TRACERED utility.

Designating Entries to Be Captured or Filtered

Although trace tables can be saved on tape or in system trace files by the CP TRSAVE command, the rate at which trace entries are generated may exceed I/O capabilities. In such situations, you can filter out certain entries. The goal is to capture only the trace information of interest.

Use the CP SET CPTRACE command to disable as many trace codes as possible, while still maintaining the necessary history of system events.

To designate which entries are either captured and written to a trace table or filtered out and not written to a trace table, specify the following:

1. Trace codes
2. User ID or SYSTEM.

   Note: SYSTEM represents the trace entries CP creates while doing work for the system. This includes all work dispatched on the SYSTEM VMDBK for serialization.

Capturing or Filtering Data by Trace Code: If you want to capture or filter data for certain trace codes, use the CP SET CPTRACE command to trace individual codes or named categories of codes.

Capturing or Filtering Data by User ID or SYSTEM: In addition to designating trace codes for capturing or filtering, you can further limit the trace entries written to trace tables by designating other tracing criteria. These additional tracing criteria include user ID, SYSTEM, or certain groupings of these. Use the CP SET CPTRACE command with the SPECIFIC option to designate certain user IDs be traced, each with its own set of tracing criteria. Use the CP SET CPTRACE command with the NONSPECIFIC option to designate certain user IDs be traced, all sharing the same tracing criteria.

Figure 5 illustrates the concept that you can request tracing according to separate tracing criteria for individual user IDs or shared tracing criteria for a group of user IDs.

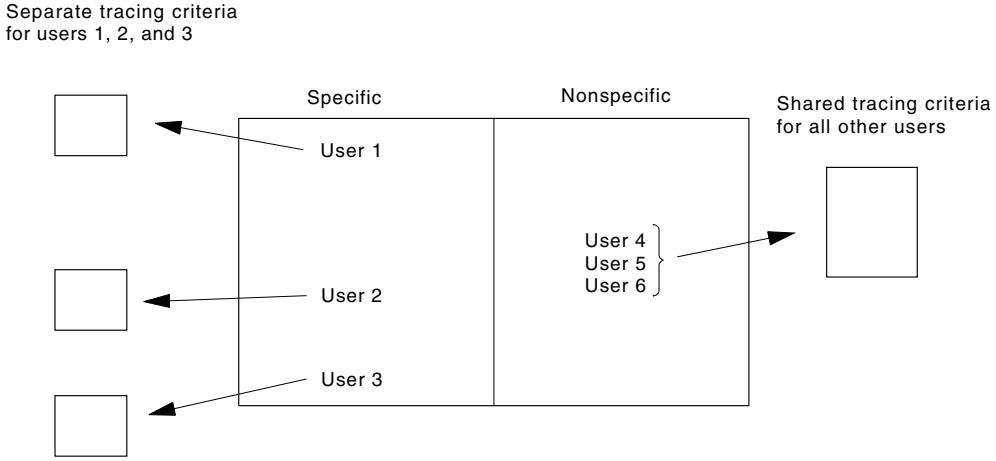


Figure 5. Tracing Events for Specific and Nonspecific Users

For additional information about the SET CPTRACE command, see the *z/VM: CP Commands and Utilities Reference*.

More Information on Filtering

The following system events are some of the most common entries in a trace table. If you do not need them for diagnosing problems in a particular circumstance, filter them out to reduce the number of trace entries generated.

System Event	Entry Code
Obtain free storage frame	CODE=0600
Return free storage frame	CODE=0700
Run user	CODE=0A00
Call with save area	CODE=2800
Return with save area	CODE=2C00
Stack CP execute block	CODE=3300
Unstack CP execute block	CODE=3310
Exit to the dispatcher	CODE=3600

Tracing I/O, Data Code Paths, and Virtual Machines

The TRSOURCE command lets you trace I/O paths, data code paths, Guest LAN or VSWITCH paths, and virtual machine guests. You can use TRSAVE to save the source data on DASD and the TRACERED utility to format the data so that you can read it interactively. The following are examples of using TRSOURCE for tracking I/O, data paths, and virtual machine guests. For an example of using TRSOURCE with a Guest LAN or VSWITCH problem, see "Using TRSOURCE to TRACE a Guest LAN or Virtual Switch" in *z/VM: Connectivity*.

I/O Trace Example

The operator gets a system message (COMMAND REJECT) indicating an I/O error on the 3800 printer at real device address 411.

To look at the CCWs to this device, enter the following two commands:

```
     trsource id printbug type io dev 411
trsource enable id printbug
```

Wait for the error to recur. At that time, enter this command:

```
trsource disable id printbug
```

You can now enter QUERY TRFILES to make sure that one or more trace files were created. The user ID that issued the TRSOURCE commands is the owner of these trace files. If you received message 6084 saying that the oldest trace file was purged, more trace data was generated than could be contained in two 256-page files. You may change the size or number of files that are created when you enable the trace ID. If you choose to specify five 400-page files, enter:

```
trsave for id printbug size 400 keep 5
```

See the TRSAVE command in *z/VM: CP Commands and Utilities Reference* for more information.

Trace Table Example

The problem

Several users are reporting that their user IDs seem to be hung because they cannot log off. This happens every day between 4:00 and 5:00 in the afternoon when they want to go home. Their user IDs are USER1, USER2, USER3, and USER4.

The research

You have taken a restart dump. In further analysis, you find that these user IDs were hung because a wait flag is being turned on but never turned off for them. The

restart dump does not reveal the cause because the trace table had wrapped by the time the dump was taken. There are no events for these users in the dump.

The solution

Between 4:00 and 5:00 P.M. tomorrow, obtain the events that occur for these users. You have two 3590 tape drives located at real device addresses 181 and 182.

At 4:00 P.M., enter this command to turn tracing off for the system and for all users.

```
set cptrace off
```

Now enter the following commands to turn tracing on for these four users :

```
set cptrace for user1 on
set cptrace for user2 on
set cptrace for user3 on
set cptrace for user4 on
trsave for cp on tape 181 182 rewind
```

At 5:00 P.M., enter:

```
trsave off
```

To start the tracing for the system and for other users again, enter:

```
set cptrace on
```

You may now use the TRACERED utility to display the trace data on the tapes.

Data Trace Example 1

When using an application that uses IUCV to transmit data, end users are complaining that they are receiving incorrect data. There are three possible points at which the incorrect data may be originating:

1. The sending (SOURCE) virtual machine
2. The CP send/receive mechanism (IUCV)
3. The receiving (SINK) virtual machine.

Step A

Understand what data is supposed to be sent from the SOURCE virtual machine.

Step B

Find out what data is actually being sent. (If this data does not match what is supposed to be sent, the SOURCE virtual machine is the origin of these problems.)

At offset X'1B2' in module HCPMOD, register 5 points to the user data; register 6 points to the control block describing the data. The instruction at this location is LR R1,R5 (X'1815').

Set up a data trace to trace the general registers, the storage pointed to by register 5 for 200 bytes, and the storage pointed to by register 6 for 100 bytes. Enter the following command:

```
trsource id send type data loc hcpmod + 1b2 1815 dl g0:f g5.200 g6.100
```

Step C

Find out what data is being received by the receiving virtual machine. If the data is the same as what was being sent, then IUCV is not the origin of the incorrect data. Otherwise, IUCV is the problem source.

At address X'2B200', data is passed to the SINK virtual machine. The instruction at this location is SLR R5,R5 (X'1F55'). Register 4 points to the user data. Register 7 contains the pointer to the control block that describes the data. Set up a data trace to trace the storage pointed to by register 4 for 200 bytes and the storage pointed to by register 7 for 100 bytes. Enter:

```
trsource id sink type data loc 2b2000 1f55 dl g4.200 g7.100
```

Step D

Collect the data. You are planning to analyze the data from a different user ID (USERB) than the one issuing the TRSOURCE commands. Therefore, use TRSAVE to change the user ID that will receive the files when the trace is completed. Enter the following three commands:

```
trsave for id send to userb
trsave for id sink to userb
trsource enable id send sink
```

Wait for the problem to occur, then enter:

```
trsource disable id send sink
```

USERB may now use the TRACERED command to process the trace data recorded by TRSOURCE.

Data Trace Example 2

The following example depicts how multiple TRSOURCE command invocations may be entered to set up a conditional data link trace.

You have been experiencing system abends and based on preliminary dump analysis you suspect an overlay is occurring. Information you've found so far in the CP Trace Table at the time of the abend leads you to suspect that the error takes place during execution of module HCPNOS.

Step A

Decide what information needs to be displayed to more closely pinpoint the error.

Step B

If appropriate, use the selectivity options of TRSOURCE when defining a conditional data link trace. The example below defines a trace at X'34' into HCPNOS at the X'58' LOAD instruction.

```
trsource id trc1 type data loc hcpnos + 76 5840C048
```

Step C

Collect the data. Because you suspect that the error occurs while the dispatched machine is either 'OPERATOR' or 'MAINT', the next two trace instructions check the VMDBK for the ID of the machine. If it is OPERATOR, then registers 0 through 15 are displayed. If it is MAINT then 48 bytes of the program header information that is pointed to by register 12 are displayed.

```
trsource id trc1 if gb+200.8 EQ C'OPERATOR'
trsource id trc1 then dl g0:f
trsource id trc1 else if gB+200.8 eq C'MAINT'
trsource id trc1      then dl gc.30
trsource id trc1      endif
trsource id trc1 endif
trsource enable id trc1
```

As with example 1 TRSAVE can be used to change the user ID that will receive the files when the trace is completed. After the data is collected the trace can be disabled.

Saving Trace Data on Tape or DASD

CP Trace table data may be saved in system trace files (TRFILEs) or on tape. Data from traces defined by TRSOURCE may be saved only in system trace files.

If the system abends while trace activity is active, the trace information that has not been recorded on DASD or tape at the time of the abend can be extracted from the CP dump by the TRSAVE subcommand of the VM Dump Tool.

Factors That Affect Saving Trace Data

Number of Trace Table Pages: CP's ability to save trace table pages before they wrap depends on the number of trace table pages available and the speed at which the entries are generated.

The number of trace table pages available to each processor is determined by the:

- Real storage size of the system (that is, by default)
- STORAGE statement in the system configuration file
- SET TRACEFRAMES command.

For more information, see the STORAGE statement description in the *z/VM: CP Planning and Administration* book and the SET TRACEFRAMES description in the *z/VM: CP Commands and Utilities Reference*.

Contention with Other Users or Functions: Trace tables are saved on tape at a lower rate of speed if other users or functions are on the same control unit as the tape drive you selected to save the trace tables.

Rate of Data Collection: If the rate of data collected exceeds the I/O rate for saving trace data on tape or DASD then some trace data may not be saved.

The DEFERIO operand of the CP TRSAVE command can be used to delay the I/O until after the trace has been turned off. With this option real storage frames are taken from the dynamic paging area and set aside to hold an in-storage wrap of the collected trace data. The oldest trace data is discarded when the wrap occurs (all frames have been filled), so enough real storage frames need to be set aside to hold the oldest trace data that you need. Filtering the amount of data collected can decrease the amount of real storage frames needed.

Trace Wrapping: When determining the amount of data that needs to be saved before wrapping, (the TRSAVE command's FRAMES parameter for in-storage wrap, SIZE, or both and KEEP parameter of wrapping of trace files on DASD), you need to consider the size of the trace records collected and the frequency of the trace events.

Options Selected on the TRSAVE Command: If you are tracing a problem that takes a long time to recreate, certain options on the TRSAVE command allow continued recording of the trace tables or data from traces defined by the TRSOURCE command, even as the tape is filled.

Selecting the use of two tape drives on the TRSAVE command is recommended to minimize loss of data. If two tape drives are specified, CP automatically switches to

the second tape drive when the tape on the first one becomes full. The operator can then mount another tape on the first drive so that it too becomes available for use should the tape on the second drive also become full. With this setup, automatic switching back and forth between two tape drives continues until the trace is complete.

In addition to specifying two tape drives, choosing either the RUN (rewind and unload) or the REWIND option further defines how the process of saving trace entries to tape proceeds. If you select RUN (the default), new tapes can be mounted and the drive made ready to accept additional trace information to provide an indefinite history. If you select REWIND, recording can continue after the tape is rewound. If writing continues to the drive, the new information will be written over existing information.

Viewing the Trace Tables

Use the TRACERED utility to format the trace entries saved onto tape or system trace files, or written to CMS files by the VM Dump Tool TRSAVE subcommand, and then view the information in a print file or CMS file. Use the TRACERED utility to select options and format the output. You can send the output to a CMS file for viewing on your virtual machine or for printing. See the *z/VM: CP Commands and Utilities Reference* book for more information on the TRACERED utility.

Factors affecting TRACE Table Pages

CMS Storage: You may encounter disk storage constraints if you select a CMS file for the output from the TRACERED utility. The more trace entries that meet the selection criteria, the larger are the storage requirements. One way to alleviate storage constraints is to designate more stringent selection criteria.

The table that follows shows the total number of trace entries TRACERED can process onto a single cylinder or its equivalent in number of blocks on the specified DASD type:

Table 4. Approximate Number of Trace Entries per Cylinder or per 1000 Blocks

CMS Minidisk Device Type	Formatted	Unformatted
3350	1666	5000
3375	1066	3200
3380	2083	6250
3390	2083	6250
FBA (1000/512-byte blocks)	1873	5620

You should also beware of creating CMS files too large for the CMS editor to accommodate. Should this occur and you still want to view the entries created, either use the COPYFILE command to break the file into manageable pieces or increase the virtual machine storage size. The alternative is to erase the CMS file and rerun the TRACERED utility with more stringent selection criteria.

Chapter 4. Creating a Dump

A dump is a record of the contents of your machine's storage at a given moment. It can appear either online or printed on paper. A dump can pinpoint the moment when malfunctions begin.

A dump can originate in a z/VM system within:

- CP
- A virtual machine in which CMS, or another z/VM component, or a guest operating system is running
- A communication controller.

A dump, depending upon the type you request and where it comes from, can include data such as:

- Virtual storage, which is a byte-by-byte record of the contents of a virtual machine's storage in hexadecimal notation. The dump provides an EBCDIC translation of this data.
- Real storage, which is a byte-by-byte record of the contents of your z/VM system's real storage and includes control blocks
- Access, general purpose, and floating-point registers
- Control registers
- The time-of-day clock
- The processor timer
- The program status words (PSWs).

Types of Dumps

There are several types of dumps you can request, depending on the information that you want.

- **A CP dump.** This is a dump of the storage directly owned by CP. It is generated by CP during a hard abend and results in system termination and possibly a restart.
- **A snapdump.** This is a dump of the storage directly owned by CP and is very similar to a hard abend dump but does not result in system termination.
- **A CP soft abend dump.** A soft abend dump is a dump of a small amount of the storage directly owned by CP. It is created when CP encounters a problem where system integrity is not jeopardized by the error, or when CP can isolate an error to a virtual machine. It does not result in system termination.
- **A stand-alone dump.** Sometimes, a problem can be so severe that your system cannot even produce a CP dump on its own. For this reason, every z/VM system is equipped with a special program that will produce a dump of real storage, regardless of how severe the problem is. It is called a **stand-alone** dump because the program that produces it stands alone or independent of the rest of the system programming. Because it is independent of the system programming, any problems there will not prevent the dump from being created.
- A dump limited to **any single virtual machine** (VMDUMP) running in your z/VM system. For example, you can request a dump of a virtual machine containing CMS, RSCS, or any guest operating system that resides in a virtual machine.
- A dump of a **communication controller's storage.** A communication controller is a device that manages and controls the operation of a computer network, including the routing of data therein. Such a device contains what is called a

communication controller program, a dump of which can be useful when dealing with computer network problems. To dump information from a communication controller, see the publication associated with the type of controller installed at your location. If you use the CP CCLOAD utility to produce a communication controller dump, you can use the CP CCDUMP utility to format the dump file. For more information, see the *z/VM: CP Commands and Utilities Reference* book.

A dump is useful when dealing with a problem in your z/VM system. A dump is a picture of the system's (or virtual machine's) storage. The problem is likely to be somewhere in the picture. Dumps are also especially helpful in dealing with wait states, infinite loops, and abends.

There may be times when a dump does not provide all the information you need. In those cases, especially if the problem is a user hang, a trace table may be helpful. See Chapter 3, "Using Traces to Debug," on page 39 for more information.

Setting Up the System for a Dump

You must route your dump to the appropriate destination and allow sufficient space for the dump.

1. Specify the appropriate dump medium and routing.

   When CP creates a dump, the dump is sent to the virtual machine defined in the SYSTEM_USERIDS statement in the system configuration file. You should use the DUMPLOAD utility to load the dump from the reader spool file into a CMS dump file.

   If you wish, you can specify in advance the destinations for the dump. Use the CP SET DUMP command to indicate where you prefer to send a dump whenever one is generated. You can specify up to eight DASD devices, or one tape. The *z/VM: CP Commands and Utilities Reference* describes the SET DUMP command in detail.

2. Provide sufficient spooling space to accommodate the dump.

   A system dump uses a significant amount of spooling space. The amount of space required depends on the amount of real storage on the processor in the real machine and the type of DASD allocated for spooling. For example, a dump from a 16 MB machine fills approximately 27 cylinders of a 3380 device. The *z/VM: CP Planning and Administration* book contains a table of suggested dump space allowances for various storage sizes and DASD types.

3. Provide sufficient minidisk space to receive the dump.

   To use the available dump viewing tools, you must process the dump into a CMS file. This requires the receiver to have sufficient minidisk space. The precise amount of space needed depends upon:

   - The amount of storage dumped
   - The type of DASD
   - The block size specified when the minidisk was formatted.

   Guidelines for storage requirements are given in the *z/VM: VM Dump Tool* book.

4. Decide which debugging tool you want to use.

   If you produce a dump of the contents of a virtual machine, consider what that machine contains. If it contains a guest operating system (such as MVS or VSE), then consider using the dump facility provided by that particular system. The quality and quantity of the data in the dump will probably be higher than that obtained using z/VM dump commands. Review the manuals pertaining to the operating system in question.

If a virtual machine contains a z/VM product or component that runs in ESA/390 Architecture mode (such as CMS or GCS), you can use the Dump Viewing Facility to view the dump.

If the virtual machine contains CP or other z/Architecture mode operating system, you can only use the VM Dump Tool to view the dumps in z/Architecture format.

Dumping Real or Virtual Machine Data

When CP abends, it automatically tries to create a dump. There may be other times, however, when you need to produce a dump. This often depends on the virtual machine running on the system.

For example, when a program you run under CMS abnormally ends, you do not automatically receive a program dump. If, after attempting to use CMS and CP to debug interactively, you still have not discovered the problem, you may want to obtain a dump.

You might also want to obtain a dump if you find that you are displaying large amounts of information, which is not practical on a terminal.

Commands That Dump Real or Virtual Machine Data

Commands that dump real or virtual machine data are: DUMP, VMDUMP, and SNAPDUMP. See the *z/VM: CP Commands and Utilities Reference* for more information on these commands.

The DUMP Command

See the description of the DUMP command in the *z/VM: CP Commands and Utilities Reference* book for a description of the real and virtual machine components that can be sent to a virtual printer.

For example, to dump the virtual storage space for a specified address range with an EBCDIC translation of the dump enter:

```
dump t20000-20810
```

See Chapter 5, “Debugging CP,” on page 55 for more information on using dumps to debug.

The SNAPDUMP Command

The SNAPDUMP command can be used to generate a full system dump identical to a CP hard abend dump without terminating the system. This type of dump is especially beneficial when debugging a "hung user" type of problem or when it is impossible to shut the system down for dump generation and analysis. The snapdump destination and dump content can be altered by the CP SET DUMP command. The SET ABEND command can be used to redefine soft abends as snapdumps. The CP DUMPLOAD utility can be used for processing dumps and the VM Dump Tool can be used for viewing dumps.

The VMDUMP Command

The VMDUMP command dumps virtual storage to the virtual card reader of a specified user ID. You should use the DUMPLOAD utility to load the dump from the reader spool file into a CMS dump file and then use the Dump Viewing Facility or VM Dump Tool to view or print it. For details, see the *z/VM: Dump Viewing Facility* and the *z/VM: VM Dump Tool* books. For a description of the format and contents of

the VMDUMP records, see “VMDUMP Records: Format and Content” on page 65. See Chapter 5, “Debugging CP,” on page 55 for more information on using dumps to debug.

To create a dump of a program you are running under CMS, you can enter the command:

```
vmdump 0-end format cms dcss
```

This example dumps all the discontiguous saved segments (DCSS) outside of the virtual machine's storage.

To dump a portion of a discontiguous saved segment, use an inline range value without specifying the DCSS option. Enter:

```
vmdump 100-25F0 format cms
```

CP dumps the contents of virtual storage from location X'100' to X'25F0', including guest storage and all the discontiguous saved segments within the specified address ranges.

Stand-alone Dump Utility

z/VM includes a stand-alone dump utility that you can tailor according to your installation's configuration using CMS. After you generate z/VM, you should create the stand-alone dump utility program and place it on tape or a DASD for emergency use. If, after a system failure, CP cannot create an abend dump, you can use the stand-alone dump utility to dump all of storage.

To use the stand-alone dump program to dump real storage, your must have access to IPL the real machine. You can IPL the stand-alone dump program from tape or a DASD and direct the output to a tape. The stand-alone dump program is not supported to IPL from FBA type DASD. You may need to reserve several tapes to hold all the information. Basic error recovery is available for DASD and tape devices used as IPL or output devices.

Typically, an installation can have several stand-alone dump programs generated and ready to run. It would be useful to have the following configurations available for the stand-alone dump utility:

- IPL from tape with output directed to tapes
- IPL from DASD with output directed to tapes

These configurations let you take a stand-alone dump with any of the supported possible environments.

See the *z/VM: CP Planning and Administration* book for information about how to create the stand-alone dump utility.

The stand alone dump program communicates with you using PSW wait codes. When the stand-alone dump program completes processing or ends because of an error, it will enter a disabled wait state and load a wait state code into the PSW. This PSW will appear on the operator's console, at the end of the wait state message you receive. In the example shown below, the wait state code in the PSW is 8200:

```
CPU STOPPED; DISABLED WAIT PSW 000A0000 00008200
```

For a description of what the disabled wait state code means, look up the CP message that has the same number as the wait state code. For the above example, wait state code 8200 indicates that the stand-alone dump has successfully completed. See the *z/VM: CP Messages and Codes* book for a description of CP messages.

See the *z/VM: System Operation* book for information about how to run the stand-alone dump utility.

Chapter 5. Debugging CP

Debugging CP in a Virtual Machine

Many CP problems can be isolated by running in a virtual machine. In most instances, the virtual machine system is an exact replica of the system running on the real machine. To set up a CP system in a virtual machine, use the same procedure that generates a CP system on a real machine. However, remember that the entire procedure of running service programs is now done on a virtual machine. Also, the virtual machine must be described in the real directory. See the *z/VM: Running Guest Operating Systems* book for directions on how to set up the virtual machine.

Abend Dumps

When an abnormal end occurs, CP attempts to dump the contents of storage. Dumps can be directed to DASD or tape.

A *soft abend dump* is taken when a problem program cannot continue, when system integrity is not jeopardized by the error, or when CP can isolate an error to a virtual machine. If the operating system for your virtual machine cannot continue, it ends and, in some cases, tries to take a dump. A virtual machine dump is sent to a system data file.

A *snapdump abend dump* is taken when a problem program cannot continue, when system integrity is not jeopardized by the error, or when CP can isolate an error to a virtual machine. Although the information contained in the snapdump is identical to that contained in a hard abend dump, the system is not terminated.

A *hard abend dump* is produced when the CP system cannot continue.

When you receive an abend, if the dump is set to go to DASD SPOOL space (specified by the CP SET DUMP command), the dump is sent to the reader of the user ID designated as the dump receiver. This user ID is specified by the DUMP operand of the SYSTEM_USERIDS statement in the system configuration file. By entering the QUERY DUMP command, you can determine where the dump is being directed. After the dump is loaded onto DASD, use the DUMPLOAD utility to create a CMS file and then use the VM Dump Tool to process it or view it interactively.

If the dump is directed to one or more tapes, use the DUMPLOAD utility to create a CMS file and then use the VM Dump Tool to view it interactively.

Use the CP SET DUMP command to designate the output device to receive system abend dumps. See the *z/VM: CP Commands and Utilities Reference* for the format of the SET DUMP command.

Reading CP Abend Dumps

When CP can no longer continue and abnormally ends, you must first determine the condition that caused the abend, and then find the cause of that condition. You should know the structure and function of CP.

Two types of dump formats occur when CP abnormally ends, depending upon where the dump is directed to in the CP SET DUMP command.

If the dump is directed to DASD, and if you want to use the VM Dump Tool to analyze it, you will need to use the DUMPLOAD utility to load the dump into a CMS file. You can then use the VM Dump Tool VMDUMPTL command to view the dump interactively. This chapter contains several references to the VMDUMPTL command. For detailed information about this command, see the *z/VM: VM Dump Tool* book.

Storage is displayed in hexadecimal notation, four words to the line, with EBCDIC translation at the right. The hexadecimal address of the first byte printed on each line is indicated at the left.

For information about obtaining detailed descriptions of CP data areas and control blocks, see "Looking at Key Control Blocks" on page 57.

Using the Assert Facility

The Assert Facility can help detect some problems earlier in execution. This facility allows some CP modules to verify that certain conditions exist before continuing execution. If the conditions are not met, an abend or stop occurs, depending on how the facility is activated.

To turn the Assert Facility on, enter:

```
CP SET CPCHECKing ON ABEND
          or
CP SET CPCHECKing ON VMSTOP
```

For more information about setting conditions, see the *z/VM: CP Commands and Utilities Reference*.

Reading the Dump with the VM Dump Tool

The VM Dump Tool gives you the ability to interactively view CP, stand-alone, soft abend, and virtual machine dumps. It runs under CMS.

To use the VM Dump Tool for diagnosing CP problems, you need the following:

- A copy of the dump you want to examine.
- A copy of the DUMPLOAD utility which you use to load the dump into a CMS file in order for the resulting dump to be usable by the VM Dump Tool.

When you receive an abend, if the dump is set to go to DASD SPOOL space (specified by the CP SET DUMP command), the dump is sent to the reader of the user ID designated as the dump receiver. This user ID is specified by the DUMP operand of the SYSTEM_USERIDS statement in the system configuration file. For information on the CP SET DUMP command, see the *z/VM: CP Commands and Utilities Reference*. For information on setting up the system abend dump environment, see the *z/VM: System Operation* book.

To use the dump with the VM Dump Tool, you must:

1. Log onto the dump receiver's user ID.
2. Load the dump into a CMS file, using the DUMPLOAD utility. See the *z/VM: CP Commands and Utilities Reference* for additional information on the DUMPLOAD utility.

The VM Dump Tool shortens the time you need to gather information about a CP problem. Some of the tasks that the VM Dump Tool performs are:

- Displaying symptom record information. By using the SYMPTOM subcommand of VMDUMPTL, you can easily check the symptom record issued with the abend dump.
- Viewing the contents of all registers and all PSW values at the time of the dump. The REGS subcommand of VMDUMPTL enables you to view the contents of general purpose, control, access, and floating-point registers, and all the PSW values at the time of the dump.
- Formatting trace entries. By using the TRACE subcommand of VMDUMPTL, you can format the trace entry so that each field of a trace entry is displayed with its description.
- Locating the addresses of certain modules or entry points in a CP dump, or identifying which modules or entry points reside at a specific address in a CP dump. Use the MAP subcommand of VMDUMPTL to do this.
- Finding real and virtual device information. The RDEVBK and VDEVBK subcommands of VMDUMPTL enable you to locate RDEVs and VDEVs by going through the radix tree. These subcommands display the data on your screen.
- Finding information about any control block. Use the BLOCK and CHAIN subcommands of VMDUMPTL to do this.

Printing Dump Information from the VM Dump Tool

After you have processed the dump so that the VM Dump Tool can use it, you can display information from the dump.

After you have completed your VM Dump Tool session, use the FILE or SAVE subcommands to save the DUMPLOG file to disk. You can use the CMS PRINT command to print this DUMPLOG file. For more information on PRINT, see the *z/VM: CMS Commands and Utilities Reference*.

Looking at Key Control Blocks

z/VM CP uses control blocks to hold information about many aspects of the entire system. System processing relies on this information so that if incorrect data is placed in these control blocks, errors occur.

When errors occur, control blocks often provide the best information about the causes. By examining the fields within the control blocks and the available source listings, you can obtain valuable diagnostic information for problems with z/VM.

Descriptions of some major control blocks appear in the following sections. For each control block, a brief explanation of its purpose is given, followed by pointers or other methods for locating the control block, and then by specific fields that you may find useful in gathering data. Although these control blocks are especially helpful in diagnosing problems, they are not the only ones you should use.

You can obtain a detailed description of CP data areas and control blocks in several ways:

- Use the VM Dump Tool BLOCK subcommand.

  The BLOCK subcommand of the VM Dump Tool can be used to format CP control blocks for displaying. See the *z/VM: VM Dump Tool* book for information about the BLOCK subcommand.
- Use the z/VM on-line control block data base.

  You may also refer to the following URL for a description of the control blocks: *www.ibm.com/eserver/zseries/zvm/library/*

HCPPFXPG: The Prefix Page

The prefix page is actually two 4K pages for each processor running in z/VM. Each prefix page contains both hardware and software information for its processor. At system generation, HCPLOD defines the IPL processor's prefix page location. Or, if an alternate processor is either brought online during IPL processing or varied online after the IPL is complete, the prefix page is acquired dynamically and its location is defined by HCPMPS.

If you receive an abend dump, you can find the address of the prefix page by using the CPUID, CREGS, or REGS subcommands of VMDUMPTL.

HCPPFXPG contains information you will find helpful in performing diagnosis. It contains the following:

- PSW information.

  The system PSWs for the processor include PGM, MCH, I/O, RESTART, SVC, and EXT.
- Linkage save areas:

  These include the following:

  PFXTMPSV A copy of the registers and the work areas when one module calls another.

  PFXBALSV The BALR linkage save area.

  PFXWRKSV The special work save area.

  PFXFRESV The HCPFRE save area.

  PFXPTRSV The page translation save area.

  PFXLNKSV The call return linkage save area.

  PFXIOSID The subchannel number of the last I/O device from which an interrupt was received, the first halfword always contains 0001.

  PFXINPRM The address of the RDEV of the last I/O device from which an interrupt was received.

  PFXRNUSR The address of the last run VMDBK.

  PFXNXTPF If multiple processors are defined, a pointer to the next prefix area.

  PFXTTPNT A pointer to the beginning of the trace table associated with this prefix page.

  PFXSYSVM The address of the system VMDBK that is the starting point of the global cyclic list.

  PFXSYS The address of the system common area (SYSCM).

HCPSYSCM: The System Common Area

The system common area, SYSCM, contains pointers, variables, counters, and constants for the entire system. It is created at system generation as part of HCPSYS. SYSCM is located by the pointer PFXSYS from any prefix page.

Diagnosis information found in HCPSYSCM includes:

SYSPRFIX The prefix area for the IPLed processor

SYSTOD The first half of the time-of-day (TOD) clock at IPL time

SYSRDEV The address of the first RDEV block in the radix tree

SYSTORS Real machine storage size up to 2 GB

SYSGTORS Real machine storage size including storage above 2 GB

HCPVMDBK: The Virtual Machine Descriptor Block

The HCPVMDBK, or VMDBK, is a control block that exists for each virtual machine that is logged on. Each descriptor block contains information about its virtual machine. It is created when a user does any of the following:

- Logs onto z/VM
- Defines an additional virtual processor
- Enters a SIE (Start Interpretive-Execution) instruction.

Each user has one VMDBK per virtual processor and one additional VMDBK for each virtual processor from which the SIE instruction was entered.

Locating Descriptor Blocks from a Dump

You can locate VMDBKs in several ways. To display a list of all the VMDBKs in a dump or to display a summary of VMDBKs for a specific user, use the VMDSCAN, VMDBK, or VMDBKS subcommands of the VM Dump Tool VMDUMPTL command. For a complete description of these subcommands, see the *z/VM: VM Dump Tool* book.

The following fields in other control blocks may be helpful to you when examining the VMDBKs:

VDEVUSER of HCPVDEV A pointer to a user's VMDBK from a virtual device accessed by that user.

RDEVUSER of HCPRDEV A pointer to a user's VMDBK from the real device owned by that user.

PFXSYSVM of HCPPFXPG A pointer to the system VMDBK from the prefix page.

You can also locate VMDBKs by chains called the global and local cyclic lists. A global cyclic list is a chain of all origin VMDBKs for users logged on. The VMDCYCLE field in the system VMDBK control block points to the first VMDBK in the list of logged-on users. Then the VMDCYCLE field of each user's VMDBK points to the next VMDBK in the global cyclic list, and on down the chain until the last VMDBK. The last VMDBK does not point back to the system VMDBK control block, but to the first VMDBK in the list, the same one to which the system VMDBK points.

To point to the primary VMDBK for a specific user in a dump, use the VMDBK subcommand of VMDUMTPL. Enter:

vmdbk *userid*

A local cyclic list is a chain of all VMDBKs with the anchor at a VMDBK in the global cyclic list. The VMDLCYCL field points to the next VMDBK on a local cyclic list. The last VMDBK on a local cyclic list points back to the origin VMDBK–the VMDBK on the global cyclic list.

To display a list of all the VMDBKs in a dump, use the VMDBK subcommand of VMDUMPTL. Enter:

vmdbks

The following fields are generally useful in gathering diagnostic information about a VMDBK:

VMDSTATE The scheduler and dispatcher state of the user. It tells you whether this user is ready to be dispatched or is idle.

VMDSLIST A description of the scheduling list of this user. This byte tells you whether this user is currently in the dispatch list, eligible list, dormant list, or not in any of the lists.

VMDDLCTL A description of the status of the user in the dispatch list. This byte gives information about the time-slice of the user on the dispatch list and whether the user should be dropped or reordered.

VMDIOACT The number of I/O operations outstanding for this user at the time the dump was produced.

VMDCFCTL A byte describing the status of the console function for this user at the time the dump was produced.

VMDCYCLE A pointer to the next VMDBK of the global cyclic list of logged-on users.

VMDLCYCL A pointer to the next user-defined or system-generated VMDBK for the user on the local cyclic list.

VMDCHRDN The anchor for the radix tree to VDEVs by device number.

VMDCHRSN The anchor for the radix tree to VDEVs by subchannel number.

HCPRDEV: The Real Device Control Block

HCPRDEV, or RDEV, is a control block that describes a device. CP uses these blocks to manage real and logical devices. There is a real device block for each real device in the system. An RDEV is also created to represent each logical device and is deleted when the logical device is no longer needed.

There are several ways to display the RDEV for a real device when reading a dump:

- Use the VMDUMPTL command of the VM Dump Tool.
  - Use the BLOCK subcommand to format and display RDEV blocks within a dump.
  - Use the RDEVBK subcommand to display summary information about real I/O control blocks. This subcommand uses a radix tree, which is described in "Using a Radix Tree Structure to Locate RDEVs."
- Follow one of the radix tree procedures described in the following text.

Using a Radix Tree Structure to Locate RDEVs

RDEVs for real and logical devices are stored in a radix tree structure. You can use information about the radix tree structure to locate RDEVs for both real and logical devices. The procedures for locating RDEVs for real and logical devices are nearly identical. In the examples that follow, Figure 6 on page 61, assume you are trying to locate the RDEV for real device 0191.

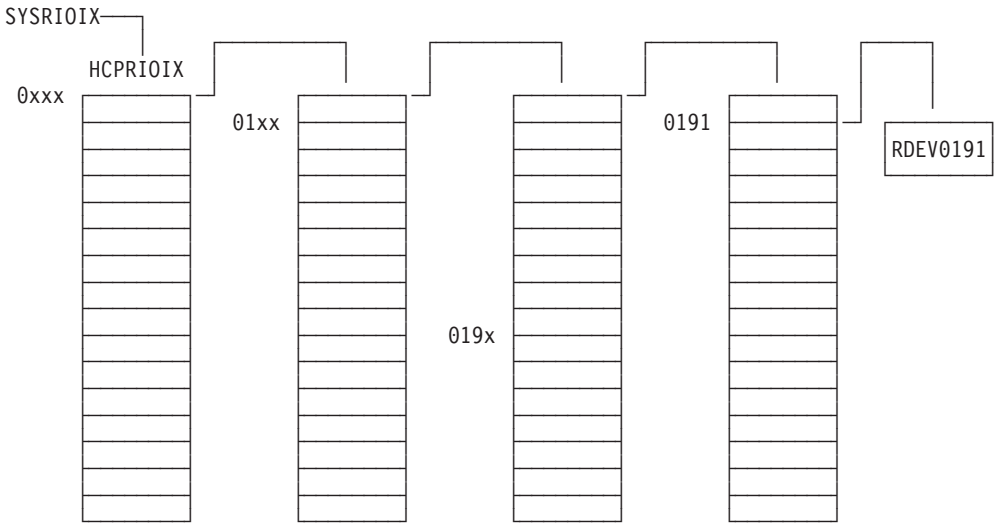


Figure 6. Using a Radix Tree to Locate an RDEV Block

Example 1: Use the VMDUMPTL command of the VM Dump Tool. When you are using VMDUMPTL to view a dump, enter:

```
rdevbk 191
```

The information you receive will point to the address of the RDEV of real device 0191.

Example 2: Use the device number in the process outlined here. The instructions below tell how to locate an RDEV for a real device. The differences in the process for locating the RDEV for a logical device are pointed out.

Note: On a running system, you can use the LOCATE RDEV command to find the addresses of a real device block and its associated control blocks.

1. Look in HCPPFXPG, the prefix page, to find PFXSYS. PFXSYS points to HCPSYSCM, the system common area.
2. Find the address of the anchor, SYSRIOIX, for the radix tree. Assume you are using a chaining procedure.
3. Look 0 fullwords past that anchor (HCPRIOIX) address because the first digit of the device number is 0.

   The address at 0 fullwords past the anchor is the next (second) address you use.
4. Look 1 fullword past that second address because the second digit of the device number is 1.

   The address at 1 fullword past the second address is the next (third) address you use.
5. Repeat this procedure for the remaining two digits, 9 and 1, for the device. The last address points to the address for the RDEV of real device 0191.

To find the logical RDEV for a logical device, use the procedure outlined above, with the following exceptions:

- Look for SYSDVFLX (rather than SYSDVFRX) in HCPSYSCM.
- SYSDVFLX points to HCPLSOLX.

- HCPLSOLX points *directly* to the first table on the radix tree (rather than to another field that in turn points to the first table on the tree).

Note: On a running system, you can use the LOCATE LDEV command to find the addresses of a system logical device block and its associated control blocks.

Control block fields of diagnostic value in the RDEV are as follows:

RDEVAIOR A pointer to the active IORBK for this device.

RDEVAFLG The control flag for the device allocated at the address of this RDEV. It describes the use of the RDEV—for instance system use, CP volume, and other usages.

RDEVDFLG The device-dependent status flag.

RDEVRFLG The device error recovery control flag.

RDEVSTAT The device-operation control flag.

RDEVVDEV The address of the VDEV, if one is present, associated with this RDEV. RDEVVDEV contains a VDEV address only if the virtual device is dedicated. When it does not contain a VDEV address, it contains zeros. If two or more virtual devices are linked to this RDEV, the address of the pointer to the VDEV addresses resides in RDEVMDSK.

RDEVMDSK The address of the MDISK block chain. The chain may consist of one or more MDISK blocks. Each block points to a chain of one or more VDEVs linked to that minidisk for a virtual machine. When RDEVMDSK does not contain the address of the MDISK block chain, it contains zeros.

RDEVUSER The address of the owning VMDBK for this device.

HCPIORBK: The I/O Request and Response Block

CP creates an IORBK whenever it needs to perform an I/O operation. When the operation completes, the IORBK is deleted.

Some key areas of the IORBK are as follows:

IORSCHED The scheduling and control flags for the IORBK.

IORFCTL A description of the subchannel function of the IORBK.

IORQSTAT A description of the status of the IORBK—waiting, active, or in dispatcher control.

IORETCOD The return code of the I/O operation after I/O is completed.

IORUSER The address of the VMDBK using the IORBK.

IORCPA The address of the channel program (CCWs).

IORIRA The address of second level interrupt handler (SLIH) routine.

IORFPNT The address of the next queued IORBK.

IORBPNT The address of the previously queued IORBK.

IORRDEV The address of the RDEV associated with this IORBK operation.

HCPVDEV: The Virtual Device Block

A VDEV describes the status of a real or virtual I/O device that can be accessed by a virtual machine. A VDEV defines the device to the virtual machine, whereas an RDEV defines the device to the system.

A VDEV remains active while the virtual machine is either running or disconnected. It is deleted only when the virtual machine is logged off or the virtual device is detached. VDEVs are created and deleted by HCPVDB.

If z/VM is running, users with class C or E privileges can find the address of a VDEV by using the CP LOCATE command. For example, to find the VDEV for USER1's 191 disk, you enter:

```
locate user1 191
```

The VM Dump Tool also offers ways to locate a user's VDEV easily. For further information on finding virtual device blocks, see the VDEVBK subcommand of VMDUMPTL in the *z/VM: VM Dump Tool* book.

To locate VDEVs from a dump, use the following pointers. Because the VMDBK has a pointer to the radix tree, the information in "Using a Radix Tree Structure to Locate RDEVs" on page 60 may also be helpful.

RDEVVDEV The address of the VDEV, if one is present, associated with this RDEV. RDEVVDEV contains a VDEV address only if the virtual device is dedicated. When it does not contain a VDEV address, it contains zeros. If two or more virtual devices are linked to this RDEV, the address of the pointer to the VDEV addresses resides in RDEVMDSK.

RDEVMDSK The address of the MDISK block chain. The chain may consist of one more MDISK blocks. Each block points to a chain of one or more VDEVs linked to that minidisk for a virtual machine. When RDEVMDSK does not contain the address of the MDISK block chain, it contains zeros.

IORVDEV A pointer from the IORBK to the VDEV for that I/O operation.

Diagnostic information found in the VDEV includes:

VDEVUSER The address of the VMDBK that owns this VDEV.

VDEVRDEV The address of the RDEV associated with this VDEV.

HCPCPEBK: The CP Execution Block

A CPEBK represents one unit of asynchronous work. The CPEBK format is identical to either the SAVBK or the SVGBK.

HCPSAVBK and HCPSVGBK: The Save Area Block

A SAVBK is a save area, as is a SVGBK. Both blocks are used in CP, but their structures and sizes are different. Most save areas are dynamic, although some are static and reside in other blocks, such as the Prefix Page and the SSABK. The formats of the save areas and the CPEBK are identical.

When a CPEBK, SAVBK, or SVGBK is used, it contains the following:
- A caller's registers 0 through 15
- The use status of the block
- Indicators of the size and format of the block.

- Work areas.

A save area may hold the general registers in one of three formats:
- Short (32–bit)
- Long (64–bit) contiguous
- Long (64–bit) discontiguous.

The short format is used for calls between modules that use only the low-order 32 bits of the general registers. The long contiguous format is used for calls between modules that use the full 64 bits of the general registers. The long discontiguous format is used for calls from modules that use the full 64 bits of the general registers to those that use only the low-order 32 bits.

Because of these different register saving conventions, the layout of the save area is different in these three cases. The short register and long discontiguous register layouts are identical, except that the latter defines an additional area to hold the high-order halves of the general registers. This area is reserved in the short register layout. The SAVBK defines these formats of the save area. The SVGBK defines the format of the long contiguous save area.

You can use the VMDUMPTL command of the VM Dump Tool to help you debug save areas from a dump. The CPEBK subcommand formats the save areas. To locate the save areas and format them, use the FINDCPE subcommand. For more information on finding save areas, refer to the *z/VM: VM Dump Tool* book.

The following fields are helpful when you are checking CPEBKs or SAVBKs:

CPEXFPNT/CPEXBPNT, SAVEFPNT/SAVEBPNT, SVGFPNT/SVGBPNT:
The forward and backward pointers for threaded lists.

CPEXSCHC, SAVESCHC, SVGSCHC:
The stacking control field specifies what type of function the block performs.

CPEXCALC, SAVECALC, SVGCALC:
The dispatching control field specifies the status of the block.

CPEXFORM, SAVEFORM, SVGFORM:
The format field specifies the size and format of the block.

CPEXREGS, SAVEREGS/SAVEHIRG, SVGREGS:
A caller's general registers.

CPEXR1,1 SAVER11, SVGR11LO:
The VMDBK address of the user for whom the block is scheduled.

CPEXR13, SAVER13, SVGR13LO:
Generally, the previous (that is, the caller's) save area.

HCPFRMTE: The Frame Table Entry

FRMTE manages frames of real storage in z/VM. It keeps track of how each frame is currently being used and what frames of storage are currently available.

The frame table is allocated dynamically at IPL time. The frame table is never deleted during processing. The start of the frame table is located by PFXFTBL in any prefix page. For further information on finding frame table entries, see the FRAME subcommand of VMDUMPTL in the *z/VM: VM Dump Tool* book.

Important HCPFRMTE fields include the following:

FRMFPNT A forward pointer to the next frame table entry for a chained frame.

FRMBPNT A backward pointer to the previous frame table entry for a chained frame.

FRMCSWRD A fullword that has the following byte fields:

- **FRMCSB0** A description of how the frame is currently being used. For example, CP is using the frame of storage for a trace table or user page.
- **FRMCSB1** A description of the static state of the real storage frame.
- **FRMCSB2** A description of the dynamic state of the real storage frame.
- **FRMCSB3** A field used in serializing the frame state changes.

VMDUMP Records: Format and Content

When a user enters the VMDUMP command, CP dumps virtual storage of the user's virtual machine. The dump goes to the reader of the user who entered the command, unless otherwise specified. CP can store this dump in the reader spool file of any virtual machine that the user specified as an operand on the VMDUMP command.

Dumps produced by the VMDUMP command and Diagnose X'94' will have two different formats based on the architecture mode of the guest virtual machine. A vmdump of an ESA/390 mode guest, such as one running CMS, will be in ESA/390 format and only contain storage up to 2 GB. This format is the same as dumps that were produced in previous releases. DVF DUMPSCAN or VM Dump Tool can be used to analyze dumps in this format. A vmdump of a z/Architecture mode guest, such as one running z/VM CP or Linux for zSeries, will be in z/Architecture format and will include storage over 2 GB if such storage exists. Only VM Dump Tool can be used to analyze dumps in z/Architecture format.

The first logical dump record contains the symptom information. The second logical dump record contains the Dump File Map. Some of the later records contain the Dump File Information Record (DFIR), the Address Space Information Record (ASIBK) and the dumped storage.

CP records the storage dump sequentially starting with the lowest address dumped and ending with the highest address dumped. CP records each byte as an untranslated 8-bit binary value.

The VMDUMP command dumps virtual storage that z/VM created for the virtual machine user. VMDUMP creates a symptom record that provides the VM Dump Tool with header information to identify the owner of the dump. After DUMPLOAD creates the CMS file from the VMDUMP system data file, the DVF DUMPSCAN or the VM Dump Tool may be used to debug errors, as well as to store and maintain error information about the virtual machine.

Chapter 6. Debugging CF Service Machine Problems

This chapter describes how to gather information pertinent to debugging a Coupling Facility (CF) service machine.

A CF service machine is a special type of virtual machine that enables a sysplex environment to be defined on a z/VM system. A CF service machine runs the Coupling Facility Control Code CFCC (Licensed Internal Code). This code is not part of the z/VM product and is loaded directly from the processor controller into the CF service machine's virtual storage.

Determining the Status of the CF Service Machine

The user that is defined as the secondary user of the CF service machine can issue a limited set of regular CFCC Commands to retrieve information about the coupling environment. This is the user ID that is specified on the CONSOLE statement of the CF service machine directory definition. The following CFCC commands may be helpful to diagnose problems with the CF service machine:

- DISPLAY MODE
- DISPLAY CHPIDS
- DISPLAY RESOURCES
- DISPLAY LEVEL

If the CF service machine does not respond to these commands, the CF service machine may be hung or may have abended. Follow the steps in the next section to diagnose problems where the CF service machine is unresponsive.

Steps to Follow When CF Service Machine Abend Occurs

When the CF service machine detects a problem, it creates a dump and does not automatically restart itself. When this occurs, you should gather information about the current environment. This information will be useful for diagnosing the problem.

- Save the spooled console log of the secondary user of the CF service machine. This is the user ID that is specified on the CONSOLE statement of the CF service machine directory definition. The CF service machine may have displayed a message indicating the cause of the problem. If the console of the secondary user was not spooled, write down any messages that were sent from the CF service machine.
- Record information about the current system such as:
  - What processor model is z/VM running on?
  - Has the processor model changed recently?
  - Has the processor Licensed Internal Code been changed recently?
  - What was the system load at the time of the problem?
- Have the system operator take a CP SNAPDUMP of the system.
- Have the system operator issue the CP RESTART MSGPROC command to restart the CF service machine.

After the CF service machine restarts, record the release and service level of the CF service machine. This is displayed on the secondary console of the CF service machine during its initialization. This can also be displayed with the CFCC DISPLAY LEVEL command.

Finding the CF Service Machine Dump

When the CF service machine detects a problem, it creates a dump. The CF service machine uses the CP VMDUMP command to dump specific ranges of storage of its virtual machine. The dumps go to the reader of the CF service machine.

Processing a CF Service Machine Dump

Because the CF service machine is not set up to process dumps, you need to transfer the dump file to another virtual machine to process it.

After the dump has been transferred to your virtual machine, load the dump onto a minidisk using the DUMPLOAD utility.

To load the dump, enter:

```
dumpload
```

See the *z/VM: CP Commands and Utilities Reference* for more information about the DUMPLOAD utility.

Diagnosing Problems for CF Service Machines

Problems with the CF service machine should be reported to the IBM Support Center. The support center personnel will analyze the CF service machine dump in order to determine the problem. Inform the support center if you have a CP SNAPDUMP of the system at the time of the problem.

Chapter 7. Debugging CMS

This chapter describes how to use the Conversational Monitor System (CMS) to help you debug CMS or a problem program. In addition, a CMS user can use the Control Program (CP) commands and facilities to debug. Information that is often useful in debugging is also included.

Debugging Commands

Here is a list of some of the commands useful for debugging. The most useful CP commands are:

- TRACE, which traces specific virtual machine activity and records the results on the terminal or printer.
- DISPLAY, which displays real or virtual machine data at your terminal.
- STORE, which alters real or virtual machine data.
- VMDUMP, which dumps virtual storage in a different format than the DUMP command. You can process the output produced by VMDUMP by using the Dump Viewing Facility.
- DUMP, which dumps real or virtual machine data at the printer.

In addition, you may also find the SET EMSG, SET IMSG, and SET WNG commands helpful for debugging. These commands control the display of error message handling, certain informational responses, and WARNING command messages.

The CMS commands described in this chapter are:

- SVCTRACE, which records information about supervisor calls (SVC) occurring in a virtual machine. When the trace is ended, the information recorded up to that point is spooled to the virtual printer.

  The use of this command is described in more detail in “Using the SVCTRACE command” on page 70.
- SET AUTODUMP, which controls the creation of an automatic dump if an abend occurs. The automatic dump can contain:
  - The DMSNUC area of CMS, the storage management work area, the page allocation table, and the loader tables
  - A dump of the entire virtual machine and any discontiguous saved segments in use.

  The use of this command is described in more detail in “Generating CMS Abend Dumps” on page 78. QUERY AUTODUMP returns the current setting of the SET AUTODUMP command.

The following CMS commands help you debug storage-related problems in your applications:

- STDEBUG, which traces the obtain and release requests made by your application. This information is displayed on your console or written to a unit record device. The trace information includes:
  - The number of bytes obtained or released
  - The address of storage obtained or released
  - The name of the subpool that owns the storage
  - The address of the caller to storage management.

- STORMAP, which provides storage information about your virtual machine. This information is displayed on your console or written to a file. If you want to see what is displayed, issue STORMAP CALL. The information may include:
  - The total blocks of unallocated storage below and above the 16 MB line
  - The size of the largest block of unallocated storage below and above the 16 MB line
  - The name of the subpool that owns the storage
  - The start address of the block of storage
  - The end address of the block of storage
  - The number of bytes of the block of storage
  - The number of pages of the block of storage
  - The storage protection key of the page in which the block resides
  - Storage attributes.
- SUBPMAP, which provides storage allocation information for subpools in your virtual machine. This information is displayed on your console or written to a file. The information may include:
  - The name of the subpool
  - The storage protection key of the page
  - The address of the subpool descriptor block
  - The number of fully allocated pages
  - The number of partially allocated pages
  - Storage attributes.

In addition, several CMS commands produce or print load maps. These load maps are often used to locate storage areas while debugging programs.

Using the SVCTRACE command

If your program issues many SVCs, you may not get all the information you need using the CP TRACE command. The SVCTRACE command is a CMS command that provides detailed information about all SVCs processed by your program, including:

- Register contents before and after the SVC
- Name of the called routine and the location from which it was called
- Contents of the parameter list passed to the SVC.

See the *z/VM: CMS Commands and Utilities Reference* for the format of the SVCTRACE command.

The SVCTRACE command has only two operands, ON and OFF, to begin and end tracing. SVCTRACE information can be directed only to the printer so you do not receive trace information at the terminal.

Because the SVCTRACE command can only be entered from the CMS environment, you must use the immediate commands SO (suspend tracing) or HO (halt tracing) if you want tracing to stop while a program is running. Use the immediate command RO to resume tracing.

Because the CMS system is *SVC-driven*, this debugging technique can be useful, especially, when you are debugging CMS programs. For more information on writing programs to run in CMS, see the *z/VM: CMS Application Development Guide for Assembler*.

Tracing Capabilities in EXECs

It may be helpful to trace EXECs that are used to diagnose problems. By tracing the EXEC, you can follow the running of the EXEC and see intermediate values that otherwise might not be obvious. There are two EXEC processors:

- System Product Interpreter
- EXEC 2.

The amount of information displayed while running an EXEC is controlled by an instruction. The instruction depends upon the EXEC processor you are using. To find the correct instruction, see the *z/VM: REXX/VM User's Guide* or the EXEC 2 HELP menu for more information.

You can also turn tracing on for the System Product Interpreter or EXEC 2 by entering the following CMS command:

```
set exectrac on
```

This causes the tracing bit in CMS to be turned on and allows tracing without program modification.

During interactive debug, the interpreter pauses after every instruction, allowing you to single step through the program.

Assume that you have a Restructured Extended Executor (REXX) program called STATUS EXEC, which gives you some status information. The contents of STATUS EXEC follows:

```
/* This EXEC gives user some status information. */
trace ?i
say 'User ID: ' userid()
say 'Time   : ' time()
say 'Date   : ' date('w')',' date()
exit
```

Notice the command `trace ?i`, which is the second line of the program. This command causes the program to go into interactive debug and to trace:

- All clauses before they are run
- Intermediate results during evaluation of expressions
- Substituted names.

When the STATUS EXEC is run **without** the trace command, you get a result that could look like this:

```
User ID:  GEORGEB
Time   :  09:50:54
Date   :  Thursday, 7 Apr 1993
```

When the STATUS EXEC is run **with** the trace command, you get a result that could look like this:

```
     3 *-* say 'User ID: ' userid()
       >L>   "User ID: "
       >F>   "GEORGEB"
       >O>   "User ID:  GEORGEB"
User ID:  GEORGEB
       +++ Interactive trace. TRACE OFF to end debug, ENTER to continue. +++
```

At this point, enter:

```
trace off
```

to end debug, or press Enter to continue processing, and you get a result that could look like this:

```
     4 *-* say 'Time  : ' time()
       >L>   "Time  : "
       >F>   "09:50:54"
       >O>   "Time  :  09:50:54"
Time  :  09:50:54
```

At this point, enter:

```
trace off
```

to end debug, or press Enter to continue processing, and you get a result that could look like this:

```
     5 *-* say 'Date  : ' date('w')',' date()
       >L>   "Date  : "
       >L>   "w"
       >F>   "Thursday"
       >O>   "Date  :  Thursday"
       >L>   ", "
       >O>   "Date  :  Thursday,"
       >F>   "7 Apr 1993"
       >O>   "Date  :  Thursday, 7 Apr 1993"
Date  :  Thursday, 7 Apr 1993
```

At this point, enter:

```
trace off
```

to end debug, or press Enter to continue processing, and you get a result that could look like this:

```
     6 *-* exit
```

As you can see in the previous example, the intermediate results of steps 3 through 6 of STATUS EXEC were traced, and processing stopped at each step.

The z/VM Procedures Language VM/REXX Interpreter also has a TRACE function and instruction. See *z/VM: REXX/VM Reference* for more information on using the TRACE instruction and TRACE function.

Nucleus Load Map

Each time the CMS resident nucleus is built, a nucleus load map is produced as a printer spool file by the HCP loader (HCPLDR). This occurs at the time the nucleus load deck is IPLed from the reader. Save this load map. It lists the virtual storage locations of nucleus-resident routines and work areas. Transient modules are not included in this load map. When debugging CMS, you can locate routines using this map. For information on obtaining a load map, see the *z/VM: Service Guide*.

Module Load Map

The module load map of a disk-resident command module contains the location of control sections and entry points loaded into storage. It may also contain certain messages and card images of any invalid cards or replace cards that are in the loaded files. This load map is useful in debugging.

There are two ways to get a load map:

- When loading relocatable object code into storage, make sure that the MAP option is in effect when the LOAD command is issued. Because MAP is the default option, just be sure that NOMAP is not specified. A load map is then created on the primary disk each time a LOAD command is issued. See the *z/VM: CMS Commands and Utilities Reference* for a description of the LOAD command.
- When generating the absolute image form of files already loaded into storage, make sure that the MAP option is in effect when the GENMOD command is issued. Because MAP is the default option, just be sure that NOMAP is not specified. Enter the MODMAP command to display the load map associated with the specified MODULE file on the terminal. See the *z/VM: CMS Commands and Utilities Reference* for a description of the GENMOD and MODMAP commands.

Note: The load map displayed by the MODMAP command includes the NUCON and SYSREF areas; the load map created by the LOAD command does not.

CMS Abend Processing

When CMS abnormally ends, any abend exit routines established through the ABNEXIT macro or the VMERROR or VMERRORCHILD event handlers established through EventMonitor Create receive control. These exit routines allow you to bypass CMS abend recovery and continue processing elsewhere. If no routine exists or the exit routine returns to CMS, the following error message appears on the terminal:

```
DMSABE148T System abend xxx called from vstor
```

where *xxx* is the abend code and *vstor* is the address of the instruction causing the abend. CMS then waits for a command to be entered from the terminal.

Finding the Reason for the CMS Abend

Determine the reason CMS abnormally ended. *z/VM: CP Messages and Codes* lists all the CMS abend codes, identifies the module that caused the abend, and describes the action that should be taken whenever CMS abnormally ends.

Types of CMS Abends

The types of CMS abnormal ends are:

1. Program exception

   Control is given to the DMSITP (CMS interrupt handler) routine whenever a hardware program exception occurs. When a program running on a CMS virtual machine abnormally ends (abends), you receive, at your terminal, the message:

   ```
   DMSABE141T exception exception occurred at vstor in
   routine routine
   ```

   DMSITP invokes DMSABE (the abend routine) and returns your virtual machine to the CMS environment. From the message you can determine the types of program checks (such as operation, privileged operation, execution, protection, or addressing) and, sometimes, the instruction address in your program at which the error occurred.

   Note: *routine* is the command name from the last SVC issued. This routine is not necessarily the one that had the exception but is supplied to indicate the last command that was running when the exception occurred.

2. ABEND macro

Control is given to the DMSSAB routine whenever a user routine processes the ABEND macro. The abend code specified in the ABEND macro appears in the abnormal end message DMSABE155T. See the *z/VM: CMS Macros and Functions Reference* for more information on the ABEND macro.

3. Halt Execution command (HX)

   Whenever the virtual machine operator signals attention and types **HX**, CMS ends and responds with `CMS`. For more information on the HX command, see the *z/VM: CMS Commands and Utilities Reference*.

4. System abend

   A CMS system routine can abnormally end by issuing the DMSABN macro. The first three hexadecimal digits of the system abend code appear in the CMS abend message, DMSABE148T. The format of the DMSABN macro is in the *z/VM: CMS Macros and Functions Reference*.

5. AbnormalEnd API

   An application may request a user or system abend through the AbnormalEnd CSL interface. This function signals a VMERROR event in the abending process, and if no recovery is performed, the VMERRORCHILD event is signaled so that a parent process can monitor when a child process is abending. See the *z/VM: CMS Application Multitasking* book for more information on the AbnormalEnd CSL routine.

Collecting Information

The following actions may be useful in determining the cause of a CMS abend:

1. Display the PSW. You can use the CP DISPLAY command to compare the PSW instruction address with the current CMS load map to determine the module that caused the abend. The CMS storage-resident nucleus routines reside in fixed storage locations.

   Also check the interruption code in the PSW.

2. Examine areas of low storage in your virtual machine.

   You can find out more about the cause of the abend from the information in the nucleus constant (NUCON) area of low storage:

   a. Examine the program old PSW (PGMOPSW) at location X'28'. Using the PSW and current CMS nucleus load map, determine the failing address.
   b. Examine the SVC old PSW (SVCOPSW) at location X'20'.
   c. Examine the external old PSW (EXTOPSW) at location X'18'. If the virtual machine operator stopped CMS, this PSW points to the instruction running when the stop request was recognized.
   d. For a machine check, examine the machine check old PSW (MCKOPSW) at location X'30'.
   e. After you have identified the module that has caused the abend, examine the specific instruction. See the source code listing if available.
   f. If you have not identified the problem at this time, take a dump by issuing the VMDUMP command.

3. Examine several other fields in NUCON to analyze the status of the CMS system. If you are using a dump, you may return to NUCON to pick up pointers to specific areas (such as pointers to file tables) or to examine other status fields. The following areas of NUCON may contain useful debugging information.

   - The save area for low storage

     This field, called LOWSAVE, is the first 160 bytes of low storage.

   - The register save area

DMSABE, the abend routine, saves the user's floating-point and general purpose registers in the following fields:

Field	Location	Contents
FPRLOG	X'160'	User floating-point registers
GPRLOG	X'180'	User general purpose registers
ECRLOG	X'1C0'	User extended control registers

- The device

  The name of the device causing the last I/O interrupt is in the DEVICE field at X'26C'.
- The last two commands or procedures processed

Field	Location	Contents
LASTCMND	X'2A0'	The last command issued from the CMS or XEDIT command line. If a command issued in a CMS EXEC abnormally ends, this field contains the name of the command. When a CMS EXEC completes, this field contains the name EXEC. EXEC 2 and System Product Interpreter do not update this field.
PREVCMND	X'2A8'	The next-to-last command issued from the CMS or XEDIT command line. If a command issued in a CMS EXEC abnormally ends, this field contains the name EXEC. When a CMS EXEC completes, this field contains the last command issued from the CMS EXEC. EXEC 2 and System Product Interpreter do not update this field.
LASTEXEC	X'2B0'	The last EXEC procedure invoked. EXEC 2 and System Product Interpreter do not update this field.
PREVEXEC	X'2B8'	The next-to-last EXEC procedure invoked. EXEC 2 and System Product Interpreter do not update this field.

- The last module loaded into free storage and the transient area

  The name of the last module loaded into free storage through a LOADMOD is in the field LASTLMOD (location X'2C0'). The name of the last module loaded into the transient area through a LOADMOD is in the field LASTTMOD (location X'2C8').
- The CMSCB chain

  The pointer to the first CMSCB is in the FCBTAB field located atX'5C0'. Each CMSCB contains simulated OS control blocks for a particular file or device and resides in free storage. The CMSCB contains a PLIST for CMS I/O functions, a simulated job file control block (JFCB), a simulated data event block (DEB), and the first in a chain of I/O Blocks (IOBs). The first fullword of each CMSCB contains a 24-bit pointer to the next CMSCB.
- The last command entered

  The last command entered from the terminal is stored in an area called CMNDLINE (X'740'), and its corresponding PLIST is stored at CMNDLIST (X'848').
- The external interrupt work area

  EXTSECT is a work area for the external interrupt handler. It contains:

  – The PSW, EXTPSW.

  - Register save areas, EXSAVE1.
  - A separate area for timer interrupts, EXSAVE.
- The I/O interrupt work area

  IOSECT is a work area for the I/O interrupt handler. The oldest and newest PSW and CSW are saved. Also, there is a register save area.
- The program check interrupt work area

  PGMSECT is a work area for the program check interrupt handler. The old PSW and the address of the register 13 save area are stored in PGMSECT.
- The SVC work area

  SVCSECT is a work area for the SVC interrupt handler. It also contains the first four register save areas assigned. The SFLAG indicates the mode of the called routine. Also, the SVC abend code, SVCAB, is located in this CSECT.
- The simulated Communications Vector Table (CVT)

  The CVT, as supported by CMS, is CVTSECT. Only the fields supported by CMS are filled in.
- The active disk table and active file table

  For file system problems, examine the active disk table (ADT) or active file table (AFT) in NUCON.

4. If monitoring a VMERROR or VMERRORCHILD event, you may retrieve event data that gives information about the abend. The data can be mapped by VNCABNH or VMASMABN macros. See the *z/VM: CMS Application Multitasking* book for more information.

A sample utility program called DACBGEN is provided on the MAINT 193 disk. This can be used to format CMS or CP control blocks into readable/printable formats. In addition to providing output that can be formatted with BookMaster™, it can also be used for customer-written control blocks that adhere to a prescribed format. Refer to the DACBGEN README file on the MAINT 193 disk for details.

Note: The output from the DACBGEN utility is z/VM product implementation information for diagnosis and must not be used for programming purposes.

Register Use

To trace control blocks and modules, it is important to know the CMS general purpose register (GPR) usage conventions:

GPR	Contents
1	The address of the PLIST
12	The program's entry point
13	The address of a 12-doubleword work area for an SVC call
14	The return address
15	The program entry point or the return code

This information should help you read a CMS dump. If it becomes necessary to trace file system control blocks, you can use the TRACE GPR command described in *z/VM: CP Commands and Utilities Reference*. With a dump, the results of the trace, and a CMS load map, you should be able to find the cause of the abend. If you choose to use a dump, the DUMPLOAD utility and the Dump Viewing Facility will help you process and use it.

Some Debugging Tips

Here are some tips for debugging after receiving a program check abend (for example, DMSABE141T):

- DMSITP, the CMS program interrupt handler, or DMSABE, the CMS abend processing module, issues error messages when a program check occurs. If a SPIE or a STAE has been issued, control is passed to the specified routine; otherwise, control passes to DMSABE to try to recover from the error. If the message DMSITP144T is issued, the UFDBUSY byte is not zero and control is halted after the message is typed. If the wait state bit is turned off in the PSW, control continues as above. Also, if the error occurred during the running of a system routine, control is halted until the wait state bit is turned off or CMS is re-IPLed.

  Note: Turning off the WAIT bit may cause damage. Use caution.
- To determine the registers and PSW at the time of the abend, get the address of PGMSECT in the nucleus constant area (NUCON X'654'). The old PSW is stored at label EPIEPSW, X'58' bytes into the DSECT. This is followed by the registers at label EPIEREGS (X'18'). The program interrupt element (PIE), needed by SPIE, primarily uses these areas. Registers 0 through 15 are stored at offset X'3C' into the DSECT. The SPIE/STAE routine or the DMSSAB routine uses the other areas within the DSECT.
- Another aid to debugging is the SVC save area (SVCSAVE) for the virtual machine. Offset X'528' in NUCON points to these areas. The save areas are easily recognizable by the check words ABCD and EFGH contained within them. The address of the SVC caller is stored at offset 4, and the name of the routine being called is saved at offset X'8'. At offset X'10', the old PSW of the caller is stored, and offsets X'18' and X'1C' hold the addresses for the normal return and the error return, respectively.

  Registers 0 through 15 are stored at offset X'20', followed by the floating-point register at X'60'. After the first check word (ABCD), the address of the next SVCSAVE area is stored, followed by the address of the previous SVCSAVE area and the address of the user's area. If the address of the next or previous SVCSAVE area is zero, the chain is ended.

  Access registers 0 through 15 are stored at offset X'D4'.

Using CMS to Debug

After an abend, you can use CMS to debug the problem. When the information provided by the abend message does not immediately identify the problem in your program, or if you think the debugging facilities of CMS are not appropriate, you should begin debugging procedures using z/VM. For instructions on how to use the CP commands, see "Commands That Monitor Events" on page 29. If you choose to produce a dump to help you debug the problem, see "Reading CMS Abend Dumps" on page 79 for information on reading a CMS dump. If you can reproduce the problem, you can use the Dump Viewing Facility to process the dump or look at the trace table.

The most common problem you might encounter is an abnormal end resulting from a program interruption.

Sometimes the information provided by the abend message is enough for you to correct the error in your source program, recompile it, and attempt to run it again.

If a CMS command is now issued, the abend routine, DMSABE, performs abend recovery and then passes control to the DMSINT routine to process the command just entered.

Setting Machines to Automatically Create Dumps

Generating CMS Abend Dumps

By using the SET AUTODUMP command, you can automatically generate a dump of your entire virtual machine or selected parts of it whenever a CMS abend occurs. You can create a dump for irrecoverable CMS system abends for all abends that occur in your virtual machine, or you can choose not to create a dump automatically.

When you use the SET AUTODUMP command, you can generate a dump containing the DMSNUC area of CMS, the storage management work area, the page allocation table, and the loader tables.

SET AUTODUMP CMS generates a dump for the following system errors:

- Program checks within nucleus resident modules
- Irrecoverable errors in the file system
- Irrecoverable storage management errors
- All other errors that result in a disabled wait PSW.

SET AUTODUMP CMS is the default.

SET AUTODUMP ALL dumps storage for all abends in the virtual machine. In addition to the abend conditions stated above, SET AUTODUMP ALL dumps storage for:

- All program checks
- The use of the ABEND macro
- The use of the DMSABN macro.

The SET AUTODUMP CMS ENTIREVM and SET AUTODUMP ALL ENTIREVM commands dump your entire virtual machine, all the discontiguous saved segments (DCSSs) currently in use, and data spaces that contain server data (in CP format).

If you do not want to create dumps automatically, you can turn AUTODUMP off using SET AUTODUMP OFF.

If you are unsure of the setting of AUTODUMP, enter the QUERY AUTODUMP command for the current setting of your virtual machine.

If you have set AUTODUMP to ALL or CMS, the dump is produced using the CP VMDUMP command. The dump is sent to the reader of the virtual machine that abended. This user also receives a message saying that the dump has been taken. For more information on the SET AUTODUMP and QUERY AUTODUMP commands, see the *z/VM: CMS Commands and Utilities Reference*.

You can use the DUMPLOAD utility to process the dump and the DUMPSCAN command (CMSPOINT subcommand) of the Dump Viewing Facility to view it. For more information on the DUMPLOAD utility see the *z/VM: CP Commands and Utilities Reference*; for more information on the DUMPSCAN command, see the *z/VM: Dump Viewing Facility* book.

Reading CMS Abend Dumps

If you want to produce an abend dump when CMS abnormally ends, enter:

```
#cp vmdump 0-end format cms dss
```

By issuing this command, a dump spool file is created and sent to your reader. Re-IPL CMS and use the DUMPLOAD utility to format the dump into a usable form. The dump formats and prints:

- Access registers
- General purpose registers (GPRs)
- Extended control registers
- Floating-point registers
- Storage boundaries with their corresponding storage protect key
- Current PSW
- Selected storage.

Storage is printed in hexadecimal representation, eight words to the line, with EBCDIC translation at the right. The hexadecimal storage address corresponding to the first byte of each line is printed at the left.

When CMS can no longer continue, it abnormally ends. To debug CMS, first determine the condition that caused the abend and then find why the condition occurred. To find the cause of a CMS problem, you must be familiar with the structure and functions of CMS. You also need a current CMS nucleus load map to analyze the dump.

Looking at Dump Errors

The CMSDUMP serviceability aid may be helpful when you are looking at CMS control blocks or free storage chains within a CMS dump. The CMSDUMP aid is shipped with z/VM version 5 release 2 on an “as is” basis, to optionally be installed on the MAINT 193 disk. The documentation for CMSDUMP comes with the serviceability aids package. For more information see the HELPXEDI CMSDUMP file that comes with the package on MAINT’s 193 disk.

Creating Dumps in Case of Messages

By using the SET TRAPMSG command, you can automatically generate a dump of your entire virtual machine or selected parts of it whenever a specific CMS message occurs. Use the SET TRAPMSG command to set a trap to spring on a particular message, and optionally, to specify how much storage to dump.

SET TRAPMSG ON must be specified with a message number or message ID parameter. Unless a range is specified, the default dump range is ‘0 to vmsize-1’.

The dump will generate a VMDUMP format spool file when the trap springs. The type of virtual machine being dumped is CMS. The dump can be viewed using the Dump Viewing Facility.

You can check whether a TRAPMSG has been set using the QUERY TRAPMSG command.

SET TRAPMSG OFF is the default setting.

For more information about the SET TRAPMSG command refer to the *z/VM: CMS Commands and Utilities Reference*.

Printing a CMS Dump File

Use the Dump Viewing Facility PRTDUMP command to print CMS dump files that were previously created with the DUMPLOAD utility. See the *z/VM: Dump Viewing Facility* book for more information on the PRTDUMP command and *z/VM: CP Commands and Utilities Reference* for more information on the DUMPLOAD utility.

Commands That Alter the Contents of Storage

You can use the STORE (Guest Storage) and STORE (Host Storage) commands to alter the contents of virtual machine storage and host storage, respectively.

You can use the ZAP and ZAPTEXT commands to alter modules, OS LOADLIBS, TEXT libraries, or TEXT decks before the code is loaded and run.

The ZAP command is described in the *z/VM: CMS Commands and Utilities Reference* and the ZAPTEXT command is described in the *z/VM: VMSES/E Introduction and Reference*. For information on the STORE (Guest Storage) and STORE (Host Storage) commands, see "Altering Contents of Virtual Machine Storage (STORE Guest Command)" on page 34 and "Altering Contents of Host Storage (STORE Host Command)" on page 35, as well as the *z/VM: CP Commands and Utilities Reference*.

Diagnosing SFS Related Application Errors

Applications and CMS commands manipulate files residing on the Shared File System in the following ways:

- Callable Services Library (CSL) Routines
- CMS File System macros
- OS Simulation macros.

The causes of SFS related errors and warnings are well defined to applications that use CSL routines, particularly when the extended error (WUERROR) parameter is included when manipulating files on SFS.

Applications that use CMS file system or OS macros to manipulate SFS files may not get enough information through the defined interface to enable an application developer or system programmer to properly diagnose the cause of the error. However, the internal **DMSFSERR** trace table maintains SFS diagnostic information relating to recent errors and warnings detected by these macro services.

This table is allocated when the first SFS error or warning is detected by CMS File System macro services following an IPL of CMS. It will maintain a number of error records. That number is defined in the FVS control block, in the FVSFSSZ field.

The DMSFSERR table may be of particular benefit for intermittent errors, which are difficult to trace.

The format and contents of the DMSFSERR table as well as the other CMS control blocks referenced below are documented at the url: *www.ibm.com/eserver/zseries/zvm/library/* It can be located in storage of the virtual machine experiencing the problem as follows:

1. Find the location of the AFVS field in NUCON (NUCON is at offset 0 in the virtual machine.) The value in AFVS contains the address of the FVS control block.
2. Locate the FVSFSER field in the FVS control block.

3. If the value in the FVSFSER field is zero, there is no DMSFSERR trace table allocated. If the value is non-zero, it will point to the start of the table.
4. The value in FVSFSER contains the address of the DMSFSERR table. Within DMSFSERR, the value in FSESIZE indicates the size of the table in bytes. By finding the value in FSECURSR and backing up one entry, you can find the most recent error entry. Use the DMSFSERR macro to see how the data is arranged. Note that date and time information can help you navigate through the table. Also note that when the table is filled, it will wrap to the beginning.

In most cases the file id, file system operation name, return code, and reason code data in the FS error trace table will be sufficient to diagnose the cause of the error. These reason codes are documented in the CMS Messages and Codes documentation in the CSL Reason Code section of the *z/VM: CP Messages and Codes*.

Extended error information is available in many cases if additional diagnostic information is needed. Refer to the WUERROR and FPERROR macro descriptions in the *z/VM: CMS Macros and Functions Reference* for the layout of SFS extended error information.

The FS2SFSER sample program may be useful for displaying the contents of the DMSFSERR trace table in your virtual storage. Note that it is distributed on an “as is” basis, to be installed as a sample on the MAINT 193 disk. For more information, see the file HELP FS2SFSER.

Diagnosing CMS File System Errors

In addition to application errors, the CMS minidisk file system may detect some structural errors or irrecoverable processing failures. Some symptoms of file system errors are:

- The CMS file system detects an irrecoverable error. This is accompanied by a DMS1307T message; the system is placed in a disabled wait, and a VM dump is generated.
- Files cannot be read from minidisk.
- A CMS formatted minidisk cannot be accessed (device error on access).
- Duplicate files appear on a CMS minidisk.
- Files disappear from a CMS minidisk.

Some file integrity errors may be temporary, for example, when a disk is accessed read only and there are updates to the disk. However, other errors may be indications that the minidisk has been damaged. While minidisk corruption may be due to any number of factors, some of the more frequent include:

- Hardware I/O errors
- Invalid configuration of minidisks (for example, overlapping minidisk extents in the CP directory or allocation of the real pack not in PERM space.)
- Multiwrite links to a CMS formatted minidisk
- Applications that modify the virtual device addresses or links of minidisks accessed in R/W mode without releasing the file mode .
- File mode 6 (update-in-place) files open for output during a system crash
- Minidisk caching of shared DASD
- Release of storage that is critical to the file system
- Overlays of critical file system storage
- Timing errors

- Use of undocumented file system interfaces or control blocks by applications or IBM program products.
- Storage management chain corruption
- VM/CMS system errors.

Notes:

1. CMS minidisk corruption may be experienced as a side-effect of other system outages or failures, some that appear to be unrelated. It is therefore recommended that you examine any EREP data when there is minidisk corruption.
2. Minidisk corruption may be detected a long time after the minidisk was actually corrupted. For example, when a file block is marked as belonging to two different files (via pointer blocks or FSTs), frequently the error is first detected when the second file is erased.
3. After any corruption of a minidisk has been detected, it is recommended that all files unaffected by the corruption be copied to another disk, the corrupted disk be reformatted, and data be copied back to the newly formatted disk. Otherwise additional latent disk corruption may surface.
4. If I/O errors are present, you may wish to attempt to move the minidisk to another physical pack.
5. Consider using MDCHECK to analyze minidisks in which corruption is suspected. MDCHECK is an optionally installed diagnostic aid available on the MAINT 193 disk. See “Diagnosis Tools Available” on page 83.
6. When an error appears to be caused by a CP I/O error, return information may be available in DIOSECT.
7. If a storage overlay is suspected, examine some of the following data areas:
   a. Active Disk Table (ADT)
   b. Active File Table (AFT) if one exists
   c. Device Table (DEVTAB) for the affected device.

Diagnosing Data Compression Services System Errors

When using Data Compression Services to compress your data, you will be building both a compression and expansion dictionary on your A-disk. You must set up your A-disk as a read/write disk and allocate enough space so that the output files generated by the CSRBDICV EXEC will execute correctly. Messages that are requested with the *msglevel* argument will also be written to your A-disk. If sufficient space is not allocated, the output will be incomplete and unpredictable results will occur.

If you get the message

```
colaps must be X, L, AM, or AAM
```

when you are using the CSRBDICV EXEC to build compression and expansion dictionaries, you must:

1. Check that one of the valid values has been entered in the *colaps* argument positional offset
2. Ensure that there are no sequence numbers in the far right columns of the SPECFILE data which would offset the positional specifications. If they are present, delete the sequence numbers.

If you are using the CSRBDICV EXEC and a REXX fixed point overflow error occurs, you must:

1. Check that the SPECFILE data has been accessed correctly. The *scanfilename* BDICT*sf* file will contain the SPECFILE data image read during processing.
2. Check that the *maxnodes* value which has been entered is large enough to account for at least the base number of nodes in each size of dictionary.

   Note: To ensure your *maxnodes* is always set correctly, do not set it for less than the dictionary size number of entries. For example, a .5K dictionary should have a *maxnodes* of at least 512; a 1K dictionary should have a *maxnodes* of least 1024.
3. Check if the scan data is unusually large or the stepping argument of the SCANFILE forces most lines to be hit in one pass. If this exists, then either:
   - Increase the *maxnodes* value to a number near the value returned by the TN argument for the *maxnodes* approximation, or
   - Adjust the *stepping* argument value to hit fewer lines per pass.

After expanding a string of data, you may notice unexpected characters at the end of the string. To correct this, you must check the CMPSC_BITNUM bit in the CMPSC_DICTADDR_BYTE3 field of the CSRYCMPS area after a call to Data Compression Services. If this bit is on, you must add 1 to the length of the source area before calling Data Compression Services to expand your data. To test this bit, use a TM instruction.

CMS OS/VSAM users can find error code information in the “OS/VSAM Error Codes” section of the *z/VM: CMS Application Development Guide for Assembler* for OPEN, CLOSE, and I/O Request error code tables.

For more information on VSE/VSAM Data Compression Services, see *VSE/VSAM Version 6 Release 1 Commands*, *VSE/VSAM Version 6 Release 1 User’s Guide and Application Programming*, and *VSE/ESA Version 2 Release 1 Messages and Codes*.

When Calling IBM Software Support

If the problem persists, and you are unable to determine the cause of the problem, contact your IBM software support center. The following information will be of help when diagnosing the problem:

1. System dumps are generated by DMSDKD1307T error messages when the file system detects an irrecoverable error, unless SET AUTODUMP has been set to off. These should be retained to analyze the problem if needed.
2. Make a copy of the affected minidisk as soon as possible after minidisk corruption has been detected using the CP DASD Dump Restore (DDR) utility.

Diagnosis Tools Available

The following diagnosis aids may be useful in assisting you to diagnose file system failures. These are provided on an “as is” basis, to be installed as samples on the MAINT 193 disk.

AFTCHAIN may be used to determine what files are currently open, and optionally display or format Active File Table entries associated with each open CMS file.

MDCHECK may be used to validate the integrity of a CMS minidisk, and optionally recover most of its contents. Note that when MDCHECK is first run against a minidisk, pre-existing (or latent) disk corruption may be detected.

PRINTFST may be used to display the contents of a file status table (directory) entry.

PRINTBLK may be used to display the contents of a minidisk file block.

Note that documentation for these service aids is included as part of the tools themselves.

Chapter 8. Debugging CMS Pipelines

This chapter describes how to debug a problem in CMS Pipelines. This information includes techniques you can use to help isolate the problem to a particular stage of a pipeline or to a particular module in CMS Pipelines. You can then provide the information you collect to your IBM service representatives to assist them in resolving the problem.

The following sections describe debugging:

- A program exception in CMS Pipelines
- Incorrect output from CMS Pipelines
- A CMS Pipelines stall.

Debugging a Program Exception in CMS Pipelines

A program exception in CMS Pipelines may be caused by an error in CMS Pipelines or in a user program. Addressing or protection exceptions are often caused by a user program calling another program with registers set incorrectly. To isolate a program exception in CMS Pipelines, it is necessary to find the module where the error occurred and the displacement of the failing instruction in the module.

The first time CMS Pipelines is started, it installs itself as a nucleus extension. If you do not know whether CMS Pipelines has been started, enter the following command to start it and display an informational message about the version of CMS Pipelines that you have started:

```
pipe query
```

After starting CMS Pipelines, if you enter:

```
nucxmap
```

you will see results similar to this:

```
Name      Entry    Userword Origin   Bytes   Amode (Attributes)
PIPE      03E4B5F0 03F788B8 03E4B5F0 00000000   31
PIPMOD    03E48000 03F788B8 03E48000 00086948   31  SYSTEM SERVICE        IMMCMD
*PIPSYSF  00E29008 00000000 00E29000 00021658   31  SYSTEM SERVICE
*PIPPTFF  03F49026 00000000 03F49000 00001738   31  SYSTEM SERVICE
DMSEXT    00E25000 00000000 00E25000 00002478   24  SYSTEM                        PERM
DMSSEGLP  83F90398 00000000 00000000 00000000   31  SYSTEM                        PERM
SEGRSRV   03F90398 00000000 03F90398 000007C8   31  SYSTEM SERVICE                PERM
OVLYMGR   00E99000 00000000 00E99000 00001A88  ANY  SYSTEM SERVICE                PERM
NAMEFUSE  011D43A0 03F7F558 011D43A0 00000000   31  SYSTEM SERVICE
NAMEFSYS  011D43A0 03F9E4E8 011D43A0 00000000   31  SYSTEM SERVICE
Ready;
```

PIPE is the bootstrap module used to load the CMS Pipelines module. PIPMOD is the CMS Pipelines nucleus extension that contains the DMSPIPE module. From the sample results shown previously, you see that PIPMOD is located at virtual storage location 3E48000 and has a length of 86948 bytes. The address at which PIPMOD is loaded in your virtual machine may be different.

Calculating the Displacements of the Failing Module

To calculate the displacement in PIPMOD of the failing instruction, subtract the address at which PIPMOD is located from the address of the failing instruction.

To determine the name of the failing module and the displacement of the failing instruction within the module, follow these steps:

1. Enter the following command to create a file, PIPE MAP A, that contains a list of the CMS Pipelines modules sorted in order of decreasing address:

   ```
   pipe cms modmap dmspipe | strfind /FPL/ | > PIPE MAP A
   ```

   Note: If you receive a message indicating a loader table overflow, you need to increase the number of pages of storage to be used for loader tables. See the SET LDRTBLS command in the *z/VM: CMS Commands and Utilities Reference* for more information.
2. Edit the PIPE MAP A file. Obtain the address of module FPLGDTEP, which is the first module loaded in PIPMOD. It is the last module with a file name starting with FPL listed in PIPE MAP A.
3. Add the value you calculated for the displacement of the failing instruction in PIPMOD to the address of FPLGDTEP in PIPE MAP A. The sum is the address of the failing instruction relative to the addresses contained in PIPE MAP A.
4. Using PIPE MAP A, find which module contains the address of the failing instruction. You now have the name of the failing module.
5. Subtract the address shown in PIPE MAP A for the failing module from the sum you calculated in step 3. The result is the displacement of the failing instruction in the failing module.

Recreating the Problem

Before recreating the program exception, enter the following CP commands:

set run off
This makes your virtual machine stay in a stopped state while you display the contents of registers and storage.

trace prog
This causes CP to be entered as soon as the program exception occurs.

spool console * start
This creates a virtual reader spool file containing all line mode output displayed at the console.

You are now ready to recreate the problem and record the diagnostic information. When the program exception is reported by CP, follow these steps:

1. Record the address of the failing instruction displayed by CP.
2. Enter the following command to display storage just before and after the failing instruction:

   ```
   display txxxxxxxx
   ```

   where *xxxxxxxx* is the address of storage a few bytes before the failing instruction address.
3. Enter the following command to display the contents of the general purpose registers:

   ```
   display g
   ```
4. Enter the following command to display storage before the address contained in the base register, register 12:

   ```
   display txxxxxxxx.20
   ```

   where *xxxxxxxx* is about 32 bytes less than the address contained in the base register.
5. Enter the following commands:

```
begin
spool cons stop close
```

to capture the console output and place it in a spool file in your virtual reader.

6. Calculate the displacement of the failing instruction in PIPMOD.
7. Calculate the displacement of the failing module in PIPMOD and the displacement of the failing instruction in the module.

Examples

The following are examples of program exceptions in the CMS Pipelines module, DMSPIPE. To cause the program exceptions to occur, storage containing DMSPIPE is intentionally altered for the purpose of illustration. Normally, if you receive a program exception, storage has been altered unintentionally by a program error.

Example of a Protection Exception in CMS Pipelines

In this example, a ST (store) instruction at virtual storage location 03E94CE8 is altered to use a base register of 4 rather than 13. This produces a protection exception.

To determine the address at which DMSPIPE is loaded, enter:

```
nucxmap
```

```
Name      Entry    Userword Origin   Bytes   Amode (Attributes)
PIPE      03E4B5F0 03F788B8 03E4B5F0 00000000   31
PIPMOD    03E48000 03F788B8 03E48000 00086948   31  SYSTEM SERVICE        IMMCMD
*PIPSYSF  00E29008 00000000 00E29000 00021658   31  SYSTEM SERVICE
*PIPPTFF  03F49026 00000000 03F49000 00001738   31  SYSTEM SERVICE
DMSEXT    00E25000 00000000 00E25000 00002478   24  SYSTEM                      PERM
:
Ready;
```

The contents of storage at 03E94CE8 is a ST instruction in DMSPIPE. The following CP command displays storage contents at that address:

```
cp display t03E94CE8.20
R03E94CE0  05C01FEE 90DE1004 5010D008 18FD18D1 F4 *........&. ....J*
R03E94CF0  98E1F00C 18BDD20F B060C1CE D20BB050    *q.0...K..-A.K..&*
Ready;
```

The ST instruction is altered by the following CP STORE command:

```
store s03E94CE8 50104008
Store complete.
Ready;
```

The following CP command displays the altered storage contents:

```
cp display T03E94CE8.20
R03E94CE0  05C01FEE 90DE1004 50104008 18FD18D1 F6 *........&. ....J*
R03E94CF0  98E1F00C 18BDD20F B060C1CE D20BB050    *q.0...K..-A.K..&*
Ready;
```

The following commands are entered to assist in debugging:

```
set run off
trace prog
spool console * start
```

Enter the following PIPE command to recreate the problem:

```
pipe cp query time | console
 -> 03E94CE8  ST    50104008 >> 00000008    CC 2
*** 03E94CE8    PROG    0004 -> 00F3DEB0        PROTECTION
```

The following command displays the contents of the general registers:

```
display g
GPR  0 =  03F77CF0  03F6E470  03F77FEB  0000000B
GPR  4 =  00000000  03F781D8  03F77C70  00000364
GPR  8 =  03E94F09  03F781D8  03E94EC0  03F77C60
GPR 12 =  83E94CE2  03F71860  00000000  03E94CDC
```

In this example, register 15 is the base register. It points to the entry point address of the module containing the failing instruction. The following command displays the contents of storage a few bytes before the failing instruction (including the “eye catcher”):

```
D T3E94CDC.40
R03E94CD0  00000DCA C3D7E2E8 D5E34040 90ECD00C F6 *....CPSYNT  ....*
R03E94CE0  05C01FEE 90DE1004 50104008 18FD18D1    *........&. ....J*
R03E94CF0  98E1F00C 18BDD20F B060C1CE D20BB050    *q.0...K..-A.K..&*
R03E94D00  C1BE58F0 905005EF 9023B048 5040B05C    *A..0.&......& .**
R03E94D10  4180B1B8 41F0B1B8 50F0B064 50F0B06C    *.....0..&0..&0.%*
```

The following command resumes running of the PIPE command:

```
begin
FPLINX410E ABEND 000000C4 at 03E94CEC; PSW 03EC2000 83E94CEC 00040004.
FPLINX411I ... In CPSYNT; offset 00000DE4 in FPLCOM 08/21/97 17.27.
FPLINX412I ... GPR0: 03F77CF0 03F6E470 03F77FEB 0000000B.
FPLINX412I ... GPR4: 00000000 03F781D8 03F77C70 00000364.
FPLINX412I ... GPR8: 03E94F09 03F781D8 03E94EC0 03F77C60.
FPLINX412I ... GPRC: 83E94CE2 03F71860 00000000 03E94CDC.
FPLINX413I ... Store 03E94CE0: 05C01FEE 90DE1004 50104008 18FD18D1 98E1F00C.
DMSABE141T Protection exception occurred at 83E94CEC in routine PIPE
CMS
```

Note that CMS Pipelines detects the problem and issues the appropriate messages needed to isolate the problem including the name of the failing module and the displacement of the failing instruction in the module. (The displacement actually points to the instruction following the failing instruction.)

The following command captures the console output and places it in a virtual reader spool file:

```
spool console stop
Ready;
```

If CMS Pipelines had not issued messages containing the information needed to isolate the problem, the failing module name and the displacement of the failing instruction can be calculated. The following message:

```
DMSABE141T Protection exception occurred at 83E94CEC in routine PIPE
```

gives the address of the instruction following the failing instruction. The address of the failing instruction is 03E94CE8. Enter the following PIPE command to create a file, PIPE MAP A, containing a list of the CMS Pipelines modules and their corresponding addresses:

```
pipe cms modmap dmspipe | strfind /FPL/ | > PIPE MAP A
Ready;
```

The following shows a portion of PIPE MAP A:

```
FPLRAN   3E0E232
FPLCOMWR 3E0CF2A
FPLCOM   3E0CF08
FPLRVR   3E0AF10
```

The following calculations are then performed:

```
    address of failing instruction          03E94CE8
  - address of PIPMOD (from NUCXMAP)        03E48000
                                           ----------
  = displacement of instruction in PIPMOD   0004CCE8
  + address of FPLGDTEP in PIPE MAP A       03DC1000
                                           ----------
  = address of failing instruction          03E0DCE8
      relative to PIPE MAP A addresses
```

The data in PIPE MAP A shows that address 03E94CE8 is in module FPLCOM.

```
    address of failing instruction          03E0DCE8
      relative to PIPE MAP A addresses
  - address of FPLCOM                       03E0CF08
                                           ----------
  = displacement of instruction in FPLCOM   00000DE0
```

The failing module is FPLCOM and the displacement of the failing instruction in FPLCOM is 00000DE0. This information matches the information in the messages issued by CMS Pipelines.

Example of an Operation Exception in PIPMOD

In this example, an L (load address) instruction at address 03E49434 in PIPMOD is altered. This causes an operation exception.

To determine the address at which PIPMOD is loaded, enter the following command:

```
nucxmap
```

The following shows a portion of the output from `nucxmap`:

```
Name      Entry    Userword Origin   Bytes   Amode (Attributes)
:
DMSEXT    00E25000 00000000 00E25000 00002478   24  SYSTEM                      PERM
PIPMOD    03E49000 03F4F560 03E49000 00086948   31  SYSTEM SERVICE        IMMCMD
*PIPSYSF  00E29008 00000000 00E29000 00021658   31  SYSTEM SERVICE
*PIPPTFF  03F4A026 00000000 03F4A000 00001738   31  SYSTEM SERVICE
:
Ready;
```

Storage at 03E49434 contains an L instruction. The following CP STORE command changes the L instruction to load register 15 with zero:

```
store s03E49434 41F00000
Store complete.
Ready;
```

Enter the following commands to assist in debugging:

```
set run off
trace prog
spool console * start
```

Enter the PIPE command to recreate the problem:

```
pipe query level
    00000000  ????  03EC
*** 00000000    PROG    0001 -> 00F3DEB0        OPERATION
```

The following command displays the contents of the general registers:

```
display g
GPR  0 =  00000001  03F883E0  00EAEFA5  0000000C
GPR  4 =  00000000  00000000  03F87F98  03F88078
GPR  8 =  03F4C8F8  03F4F844  03F4F560  03F88000
GPR 12 =  83E49254  03F88000  83E4943A  00000000
```

The following command displays storage just before the address contained in the base register, register 12:

```
D T3E49240.20
R03E49240  C8C50000 0024C6D7 D3C9D5E7 D9D590EC F6 *HE....FPLINXRN..*
R03E49250  D00C05C0 5810D008 1FEE90DE 100418FD    *................*
```

The following command resumes running of the PIPE command:

```
begin
DMSABE141T Operation exception occurred at 80000002 in routine PIPE
```

The following command captures the console output and places it in a virtual reader spool file:

```
spool console stop
Ready;
```

In this case, CMS Pipelines does not issue any messages so calculations must be performed to determine where the error occurred. The most likely cause of an operation exception in low storage is a branch instruction with an incorrect branch address. Because branches are normally performed using register 14 as the return address, the contents of storage just before the address contained in register 14 must be examined to find the failing instruction. Storage contents at this address were previously displayed while displaying storage at the address contained in the base register.

Storage at 03E49438 contains `05EF` which is a BALR 14,15 instruction. Because register 15 contains a zero, the operation exception occurred. Therefore, the address of the failing instruction is 03E49438. Note that storage just before the BALR instruction contains the L instruction that was altered.

To calculate the name of the failing module and the displacement of the failing instruction in the module, enter the following PIPE command to create a file, PIPE MAP A, containing a list of the CMS Pipelines modules and their corresponding addresses:

```
pipe cms modmap dmspipe | locate 1.3 /FPL/ | pipe map a
Ready;
```

The following shows a portion of PIPE MAP A:

```
FPLSQIPR 03E44C0E
FPLSQIRR 03E44AF2
FPLSQIRB 03E44A02
FPLSQICR 03E448EA
FPLSQICM 03E447FA
FPLSQICN 03E48532
FPLSQIDZ 03E488E8
FPLSQL   83E425E8
FPLSPXPT 83E424B0
FPLSPX   83E3F830
```

The following calculations are then performed:

```
    address of failing instruction          03E49438
  - address of PIPMOD (from NUCXMAP)        03E49000
                                           ----------
  = displacement of instruction in PIPMOD   00000438
```

```
  + address of FPLGDTEP in PIPE MAP A       03D55000
                                           ----------
  = address of failing instruction          03D55438
      relative to PIPE MAP A addresses
```

The data in PIPE MAP A shows 03E49434 is in module FPLINX.

```
    address of failing instruction          03D55438
      relative to PIPE MAP A addresses
  - address of FPLINX                       03D55220
                                            --------
  = displacement of instruction in FPLINX   00000218
```

The failing module is FPLINX and the displacement of the failing instruction in FPLINX is 218.

Debugging Incorrect Output From CMS Pipelines

If a CMS Pipelines application fails to produce expected output or produces unexpected output, you can use the following techniques to isolate the problem:

- Add temporary stages to the pipeline to write out the data as it passes from one stage to another.
- Use the CMS Pipelines TRACE option.

Adding Temporary Stages to Write Out the Data

If a pipeline produces incorrect output, you can add temporary stages between each stage of the pipeline to write the stream to an accessed disk or directory. This helps you see what changes are made to the data as it passes from one stage to another.

Example

Suppose you receive the following output displayed on an 80-column display terminal after entering the NUCXMAP command:

```
Name      Entry    Userword Origin   Bytes   Amode (Attributes)
NAMEFIND  011D43A0 7FFFFFFF 011D43A0 00000000   31  SYSTEM SERVICE
PIPMOD    03E49000 00000000 03E49000 00086948   31  SYSTEM SERVICE        IMMCMD
*PIPPTFF  00E6E026 00000000 00E6E000 00003108   24  SYSTEM SERVICE
*PIPSYSF  00E4C008 00000000 00E4C000 00021658   31  SYSTEM SERVICE
DMSEXT    00DCB000 00000000 00DCB000 00002478   24  SYSTEM
  PERM
ERASE     03F21F3E 00E95110 03F21910 00000000   31  SYSTEM SERVICE
NAMEFUSE  011D43A0 03F80B58 011D43A0 00000000   31  SYSTEM SERVICE
NAMEFSYS  011D43A0 03F9E5D8 011D43A0 00000000   31  SYSTEM SERVICE
Ready;
```

To sort the output by nucleus extension name and to discard the heading line and the extra line containing the word `PERM`, create and start the following exec procedure:

```
/* */
'pipe cms nucxmap',
  '|nlocate string /Name/',
  '|nlocate string /PERM/',
  '|sort 1.8',
  '|console'
```

The output displayed by the exec is:

```
*PIPPTFF  00E6E026 00000000 00E6E000 00003108   24  SYSTEM SERVICE
*PIPSYSF  00E4C008 00000000 00E4C000 00021658   31  SYSTEM SERVICE
ERASE     03F21F3E 00E95110 03F21910 00000000   31  SYSTEM SERVICE
NAMEFIND  011D43A0 03F80E48 011D43A0 00000000   31  SYSTEM SERVICE
NAMEFSYS  011D43A0 03F9E5D8 011D43A0 00000000   31  SYSTEM SERVICE
NAMEFUSE  011D43A0 03F80B58 011D43A0 00000000   31  SYSTEM SERVICE
PIPMOD    03E49000 03F8F8A8 03E49000 00086948   31  SYSTEM SERVICE        IMMCMD
```

The exec sorted the output and removed the extra lines, but the information about DMSEXT was incorrectly removed. To determine whether this is a user error or CMS Pipelines error, add temporary stages to the exec to write the stream to your A-disk as follows:

```
/* */
'pipe cms nucxmap',
  '| > pipe temp1 a',
  '| nlocate string /Name/',
  '| > pipe temp2 a',
  '| nlocate string /PERM/',
  '| > pipe temp3 a',
  '| sort 1.8',
  '| > pipe temp4 a',
  '| console'
```

The file, PIPE TEMP1, reveals the problem as a user error. The line containing the string, `PERM`, is not a separate line of output. It is part of the DMSEXT line which is longer than the maximum of 80 characters the terminal can display without wrapping on to the next line. Note that the logical record length of PIPE TEMP1 is 85.

Using the CMS Pipelines TRACE Option

To generate a record of the processing of a PIPE command, use the TRACE option. The trace information generated shows what stage is being run and what data is passed from one stage to the next stage.

For more information about the TRACE option, refer to the “Debugging Pipelines” chapter in the *z/VM: CMS Pipelines User’s Guide*.

Debugging a CMS Pipelines Stall

When a stage in a pipeline cannot run for whatever reason, it is blocked. When all stages are blocked, the pipeline is stalled. A stage can be *blocked* when it has written a record to its output stream but the record has not been consumed. A record is *consumed* when a stage reads and removes the record from its input stream. A stream can also be blocked when it is waiting to read a record from its input stream, but no records are available.

When a pipeline stall is detected, CMS Pipelines displays messages describing the state of the stages when the stall occurred as well as the reason why the stall occurred. CMS Pipelines also writes a file, PIPDUMP LIST*nnnn* (where *nnnn* is a number), to your A-disk. The file contains a dump of CMS Pipelines control blocks. If the problem that caused the stall is a user error, you can erase the file. If you cannot detect the problem from the messages, it may be useful to draw a diagram of the stages, and correlate it to the stall messages. This can help determine between which two stages the problem exists. If you suspect that the stall is caused by a problem in CMS Pipelines, you can provide the dump to your IBM service representative.

Some stalls may be caused by the FANIN stage. Whenever possible, it is recommended that you use the FANINANY stage rather than FANIN. FANIN reads input records from a specified stream only. FANINANY reads input records from all streams and will not cause a set of pipelines to stall. See the *z/VM: CMS Pipelines Reference* for more information about the FANIN and FANINANY stages.

The CMS Pipelines TRACE option, which is described in "Using the CMS Pipelines TRACE Option" on page 92, can also be used to generate a trace to help isolate the problem causing the stall.

Example

The following command causes a pipeline stall:

```
pipe literal xxx | a:fanin | a:
FPLDSP029E Pipelines stalled
FPLMSG003I ... Issued from stage 2 of pipeline 1
FPLMSG001I ... Running "fanin"
FPLDSP030I Stage is in state wait out
FPLMSG003I ... Issued from stage 1 of pipeline 1
FPLMSG001I ... Running "literal xxx"
FPLDSP030I Stage is in state wait out
FPLMSG003I ... Issued from stage 2 of pipeline 1
FPLMSG001I ... Running "fanin"
Ready(-4095);
```

The pipeline stalled because there are no primary input stream records for the FANIN stage to read.

Chapter 9. Debugging the SFS Server or CRR Recovery Server

The Coordinated Resource Recovery (CRR) facility requires a CRR recovery server. The CRR recovery server functions reside in a Shared File System (SFS) file pool server, so you could have the same server performing both SFS server functions and CRR recovery server functions.

Hereafter, reference to a *server*, it could mean one of the following:

- A dedicated SFS file pool server
- A dedicated CRR recovery server
- Both an SFS file pool server and a CRR recovery server

For more information about SFS and CRR, see the *z/VM: CMS File Pool Planning, Administration, and Operation*.

The following sections describe the ways you can collect information for problem diagnosis:

- "Using the Console Log" on page 96
- "Using Server Dumps to Diagnose Problems" on page 99
- "Using System Trace Data to Diagnose Problems" on page 101
- "Using the SVCTRACE command" on page 70.

Note: The server operator does not necessarily diagnose problems, especially from the server virtual machine. Dumps and system trace data are normally used by a system programmer or whoever is responsible for diagnosing system problems.

Summary of Steps to Follow When a Server Abend Occurs

When a server abend occurs, you must follow these steps:

1. Collect information about the error.
   - Save the console log or spooled console output from the server virtual machine.
   - Save and process any dumps that the server produces.

     When an abend occurs in the server, either because the server issued an abend or because a server or CMS operation caused a program exception, the server produces a dump through the CP VMDUMP command described in the *z/VM: CP Commands and Utilities Reference*. CP sends the dump to the server's virtual reader.

     Note: The DUMP startup parameter must have already been specified in the server's DMSPARMS file to get a dump to the reader.
   - Save any system trace files that contain server data.
2. Collect other types of information about system status, such as:
   - The status of real and virtual devices that the server is using
   - The system load at the time of the error on any systems using the server and the status of each system (for example, did another system abend?)
   - The types of applications that are using the server at the time, and any information about them

- The physical connection configuration of the systems in use.

Using the Console Log

The server provides informational messages, as well as error messages, that may help you with problem determination. To keep track of the console messages, enter:

```
spool console start to userid
```

userid can be the user ID of the server virtual machine or another virtual machine user ID to whom you want the server to send the console log. You may want to add this to the server's PROFILE EXEC so a console log is always created.

To close the console log, enter:

```
spool console close
```

The log of messages received is sent to the specified user ID. See the *z/VM: CP Commands and Utilities Reference* for details on the SPOOL command.

The server provides additional information at the time of an abend to help you diagnose the problem. The console log contains information about the abend, such as:

- The abend code
- The program old PSW
- The contents of the general purpose registers.

The server also attempts to determine the displacement of the module in which the abend occurred and the displacement of the calling module.

Figures 7, 8, and 9 show some of the messages that the server may issue in response to an abend condition.

```
DMSITP141T Operation exception occurred at 4E3E72 in routine DMS5IF

SDS    ABEND SAVEAREA :

ADDR     OFFSET    DUMP DATA

0051E0C4 00000000  FFE000C1 704E3E72 00000000 003707C8  * ...A.+.........H *
0051E0D4 00000010  0051E000 0051E848 00000118 00370702  * ......Y......... *
0051E0E4 00000020  00000118 00000000 003707FC 00370760  * ................- *
0051E0F4 00000030  00372BE0 00465A60 00466A5F 00370760  * ......!-...¬...- *
0051E104 00000040  70465EE8 004E3E70                    * ..;Y.+..         *

ABTERM CODE 0C1 AT 004E3E70

PROGRAM OLD PSW IS : FFE000C1 704E3E72

GPR  0 =  00000000  003707C8  0051E000  0051E848
GPR  4 =  00000118  00370702  00000118  00000000
GPR  8 =  003707FC  00370760  00372BE0  00465A60
GPR 12 =  00466A5F  00370760  70465EE8  004E3E70

FAILURE   AT OFFSET +00058E70 IN DMSSAC  PROGRAM  (0048B000)
FAILURE   AT OFFSET +0007E410 IN  DMS3SP    89.082

CALLED  FROM OFFSET +0002E9D0 IN DMSDAC  PROGRAM  (00409000)
CALLED  FROM OFFSET +00000210 IN  DMS3RA    89.080

STORAGE NEAR FAILURE :

ADDR     OFFSET    DUMP DATA

004E3E50 00000000  00030000 0000FFFF 004FACB0 004DD958  * .........|...(R. *
004E3E60 00000010  00000080 00000000 00502330 00000000  * .........&...... *
004E3E70 00000020  00000000 10C4D4E2 F5C7D440 404040F8  * .....DMS5GM    8 *
004E3E80 00000030  F94BF0F8 F10090EC D00C18CF 5800C7DC  * 9.081.........G. *
004E3E90 00000040  58E0A014 58F0E130 5810F008 1E015500  * .....0....0..... *

POTENTIAL WILD BRANCH AT : 00465EE6

BAL(R)    AT OFFSET +0005CEE6 IN DMSDAC  PROGRAM  (00409000)
BAL(R)    AT OFFSET +00000486 IN  DMS3SP    89.082

ADDR     OFFSET    DUMP DATA

00465EC0 00000000  912F5020 90744120 91375020 90784120  * ..&.......&..... *
00465ED0 00000010  BB585020 907C5820 908C5820 201058F0  * ..&..@.........0 *
00465EE0 00000020  21D04110 906805EF 12FF4780 B4E65820  * .............W.. *
00465EF0 00000030  A2F041E0 022613EE 50E02018 1FEE06E0  * .0......&....... *
00465F00 00000040  50E0201C 50F02020 D207203C BB5040F0  * &...&0..K....& 0 *



AB/00C1           PIDS/5684-112         RIDS/DMS3SP          ADRS/0007E410
```

Figure 7. Server Console Log (Operation Exception Occurred)

```
DMSITP141T Protection exception occurred at 4FEAFE in routine DMS5IF

SDS    ABEND SAVEAREA :

ADDR     OFFSET    DUMP DATA

005390C4 00000000  FFE000C4 A04FEAFE 00000000 0038B7C8  * ...D.|.........H *
005390D4 00000010  00000000 00539848 00000118 0038B702  * ..............  *
005390E4 00000020  00000118 00000000 0038B7FC 0038B760  * ..............- *
005390F4 00000030  0038DBE0 00000000 004FEA80 0038B8A0  * .........|...... *
00539104 00000040  0038D1A0 004FEA80                    * ..J..|..        *

ABTERM CODE 0C4 AT 004FEAFA

PROGRAM OLD PSW IS : FFE000C4 A04FEAFE

GPR  0 =  00000000  0038B7C8  00000000  00539848
GPR  4 =  00000118  0038B702  00000118  00000000
GPR  8 =  0038B7FC  0038B760  0038DBE0  00000000
GPR 12 =  004FEA80  0038B8A0  0038D1A0  004FEA80

FAILURE   AT OFFSET +00058AFA IN DMSSAC  PROGRAM  (004A6000)
FAILURE   AT OFFSET +0000007A IN  DMS5GM    89.065

CALLED  FROM OFFSET +0005D03E IN DMSDAC  PROGRAM  (00424000)
CALLED  FROM OFFSET +00000486 IN  DMS3SP    89.065

STORAGE NEAR FAILURE :

ADDR     OFFSET    DUMP DATA

004FEAD8 00000000  E13047F0 C0245000 F00818B1 50D0B004  * ...0..&.0...&... *
004FEAE8 00000010  50B0D008 98F1D010 18DBD217 B0481000  * &....1....K..... *
004FEAF8 00000020  1F225020 B1405820 B0485820 20005020  * ..&.. ........&. *
004FEB08 00000030  B144183A D5063000 C7824770 C0A25890  * ....N...G....... *
004FEB18 00000040  30145820 901C47F0 C0AE182A 58902014  * .......0........ *


AB/00C4           PIDS/5684-112         RIDS/DMS5GM          ADRS/00000000
```

Figure 8. Server Console Log (Protection Exception Occurred)

```
DMS5FE3040E File pool server system error occurred - DMS4CI 05

SDS    ABEND SAVEAREA :

ADDR     OFFSET    DUMP DATA

0051D0C4 00000000  00000000 5049528E 00000010 00495A3C  * ....&.........!. *
0051D0D4 00000010  0038C000 0038C1EC 00391016 0000000D  * ......A......... *
0051D0E4 00000020  00000008 00000012 00000001 00000012  * ................ *
0051D0F4 00000030  00371340 0036FDF8 00494C90 0036FDF8  * ... ...8..<....8 *
0051D104 00000040  50495290 00500D48                    * &....&..         *

GPR  0 =  00000010  00495A3C  0038C000  0038C1EC
GPR  4 =  00391016  0000000D  00000008  00000012
GPR  8 =  00000001  00000012  00371340  0036FDF8
GPR 12 =  00494C90  0036FDF8  50495290  00500D48

FAILURE   AT OFFSET +0000B28C IN DMSSAC  PROGRAM  (0048A000)
FAILURE   AT OFFSET +000005FC IN  DMS4CI    89.058

CALLED  FROM OFFSET +0005265A IN DMSSAC  PROGRAM  (0048A000)
CALLED  FROM OFFSET +00000192 IN  DMS4SR    89.081

DMS4SB3126E SAC termination during forward processing
   LUWID = 57F5                 USERID = BRAZIE
   OPERATION = BULK INSERT
   CATALOG-ID = 6503
   PAGE-ADDRESS = 392000        PAGE-TYPE = INDEX
   PAGE-NUMBER = 112A

MS/DMS3040E       PIDS/5684-112         RIDS/DMS4CI          PRCS/05
```

Figure 9. Server Console Log (File Pool Server System Error Occurred)

Using Server Dumps to Diagnose Problems

You can use the Dump Viewing Facility to collect and diagnose problem data for the server virtual machine. The console listing, as described in “Using the Console Log” on page 96, may help you diagnose problems without using dumps.

The steps involved in using dumps to diagnose problems are:

1. Create the server dump
2. Process the server dump
3. Diagnose the server dump
4. Print the server dump.

Creating a Server Dump

The server virtual machine creates its own dumps. The dumps go to the reader of the server virtual machine. (The DUMP startup parameter must have already been specified in the server's DMSPARMS file to get a dump to the reader.) Because the server virtual machine is not set up to process dumps, you need to transfer the dump file to the appropriate virtual machine.

If the server virtual machine cannot create the dump, you can use the VMDUMP command. The VMDUMP command dumps virtual storage that z/VM creates for the virtual machine user, in this case for the server. If you enter the following CP command:

```
vmdump 0-end system format sfs
```

the dump goes to the virtual machine specified by the DUMP operand of the SYSTEM_USERIDS statement in the system configuration file. Do not use the reserved names *ATSCAB1* or *ATSCAB2* for the dump ID of VMDUMP. See the *z/VM: CP Commands and Utilities Reference* for more information on the VMDUMP command.

Processing a Server Dump

After the server virtual machine creates a dump, load the dump onto disk. To load the dump, enter:

```
dumpload
```

The default map file is SFSDVF MAP.

After you have loaded the dump, you can use the Dump Viewing Facility to format, process, view, and print the dump. To do this, enter:

dumpscan *dumpname*

When you enter the DUMPSCAN command, it checks for a server extraction routine to update the symptom record, transmit it to the symptom record repository, and update the dump.

See the *z/VM: Dump Viewing Facility* book for more information about the DUMPSCAN command, and the *z/VM: CP Commands and Utilities Reference* for more information about the DUMPLOAD utility.

Diagnosing a Server Dump

The DUMPSCAN command uses a symptom record, which is based on problem report information. The symptom record helps you find out why the server created the dump. The symptom record includes:

- Information about the system environment at the time of the dump
- The symptom string that contains the following component-related symptoms:
  - The error code
  - The ID of the failing component
  - The ID of the failing module
  - The registers and PSW contents.

To see the symptom information, use the SYMPTOM subcommand of DUMPSCAN.

You can use the other DUMPSCAN subcommands to examine the dump interactively. The following sections introduce those subcommands specifically for the server.

Formatting and Displaying Trace Records

You can scroll through the formatted output with either of the following DUMPSCAN subcommands:

- TRACE SCROLL or TRACE SCROLLU
- SCROLL or SCROLLU.

See *z/VM: Dump Viewing Facility* for more information about the DUMPSCAN TRACE and SCROLL subcommands.

Printing a Server Dump

The PRTDUMP command of the Dump Viewing Facility prints the dump and symptom record that DUMPSCAN processed. The output you get consists of the following:

- A symptom record
- A dump in hexadecimal (no special formatting)
- The contents of the registers and the PSW.

See the *z/VM: Dump Viewing Facility* book for more information on the PRTDUMP command.

Using System Trace Data to Diagnose Problems

While the server maintains an internal trace table within the server virtual machine, it also writes trace entries to the system TRFILE file. You can use the Dump Viewing Facility to format and display the trace table entries.

If you use the CP TRSOURCE command to create trace entries or the CP TRSAVE utility to save trace entries, you can format them with the CP TRACERED utility. You can then use DUMPSCAN to view server entries. For more information about the DUMPSCAN command, see the *z/VM: Dump Viewing Facility*. For information about the TRACERED utility and the TRSAVE command, see the *z/VM: CP Commands and Utilities Reference*.

Setting Internal Tracing

The server ITRACE command lets you enable or disable internal tracing for the server virtual machine. If you want to collect server trace records, enter the following from the server virtual machine after TRSAVE is started:

```
itrace on
```

If you want to stop tracing for the server, enter:

```
itrace off
```

ITRACE traces APPC/VM communications between the server machine and CMS users.

You may also start tracing, using ITRACE, by specifying the proper startup parameters when the server machine is started.

To process the internal trace output, use the Dump Viewing Facility to view the results.

A complete description of the ITRACE command is in the *z/VM: CMS File Pool Planning, Administration, and Operation* book.

Setting External Tracing

The server ETRACE command lets you enable or disable external tracing for the server virtual machine. If you want to collect server trace records, enter the following from the server virtual machine after TRSAVE is started:

```
etrace on
```

After you enter ETRACE ON, a series of prompts allow you to specify the type and level of data to be traced. The prompts you will receive are for:

- Which user ID processing will be traced. You can specify a single user ID or all user IDs with an asterisk (*).
- What type of server processing will be traced. In response to this prompt, you can specify SAC, DAC, or both to indicate the type of server processing.
- Server tracing of the subcomponents and the trace level desired.

A **0** may be entered as a response to any prompt to cancel the ETRACE command.

If you want to stop tracing for the server machine enter:

```
etrace off
```

You may also start tracing with the ETRACE command by specifying the proper startup parameters when the server machine is started.

To process the external trace output, use the Dump Viewing Facility to view the results.

When you set external tracing on, certain internal server trace records are written externally to a spool file. A complete description of the ETRACE command is in the *z/VM: CMS File Pool Planning, Administration, and Operation* book.

Other Diagnostic Facilities

There are other diagnostic aids that may be useful when working with IBM support personnel for diagnosing SFS server errors. These are distributed on an “as is” basis to be installed as a sample on the MAINT 193 disk. These include:

SFSDOT
: A set of SFS operator commands that may be useful when attempting to diagnose problems.

LCTRACE
: A facility to trace interactions between a user machine and the Shared File System (SFS). LCTRACE is invoked from a user machine’s CMS session.

Note that not all of the output formats are documented, as these are designed for IBM System Support personnel use.

Chapter 10. Debugging GCS

The Group Control System (GCS) is a multitasking operating system and is a component with z/VM. Only XA or XC virtual machines may use GCS.

XA and XC virtual machines run with the full capabilities of z/VM. Either 24-bit or 31-bit addressing can be used (thus allowing addresses below and above 16 MB), as well as the more efficient I/O using the Channel Subsystem.

While running programs on the Group Control System (GCS), you can encounter the following types of problems:
- Loops
- Abends
- Incorrect results
- Disabled wait states. [3]

To help you deal with these problems, GCS provides:
- Internal tracing facilities (see page 103)
- External tracing facilities (see page 121)
- Dumping facilities (see page 128)
- Interactive debugging support (see page 128).

Internal Tracing Facilities

The GCS supervisor maintains a wraparound trace table that serves:

- Each virtual machine individually in a group if the trace table is placed in the virtual machine's private storage
- All virtual machines collectively in a group if the trace table is placed in common storage.

The trace table is placed in private storage by default unless common storage is specified when the GROUP EXEC is run at build time. When building your GCS configuration file, you specify how big you want this table to be. The minimum you can choose is 4 KB; the maximum depends upon how much common storage you have available to use if you place the trace table in common storage. If you don't set a size limit, GCS gives you a default size of 16 KB. See *z/VM: Guide for Automated Installation and Service* for more information about how to load, build, and save GCS.

The trace table contains information about the following supervisor events:

- Task dispatches
- External interrupts
- I/O interrupts
- Program interrupts
- SVC interrupts
- I/O requests (SSCH, DIAGNOSE, HSCH, TSCH, which are called by the supervisor)
- IUCV signal system service detail entries
- SVC GETMAIN storage requests

3. Outlined in Chapter 1, "Introduction to Debugging."

- SVC FREEMAIN storage requests
- APPC/VM synchronous events
- Branch entry FREEMAIN storage requests
- Branch entry GETMAIN storage requests
- Service Point (SP) trace entries.

The tracing of supervisor events is activated as soon as your virtual machine joins a group. You can trace data from any of your GCS programs (GTRACE events) by entering the ITRACE command followed by the GTRACE macro. Service Point (SP) trace entries are activated only if you enter ITRACE SP.

Using the ITRACE Command and GTRACE Macro

To begin tracing data in a virtual machine, you must enter from the console the ITRACE command with the GTRACE option. Then the GCS application program you want to trace must call the GTRACE macro. The GTRACE macro cannot begin tracing unless you first enter the ITRACE command.

You can enter the ITRACE command for:

- Individual virtual machines
- Entire virtual machine groups.

Any virtual machine operator who enters it on behalf of the whole group (ITRACE GROUP) must have an authorized user ID.

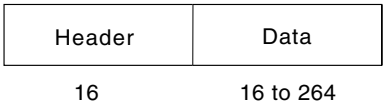
For more information about the ITRACE command and the GTRACE macro, see the *z/VM: Group Control System* book.

Note: ITRACE of GTRACE records will only trace GTRACE records that are less than or equal to 256 bytes. GTRACE records that are greather than 256 bytes and up to 8k can only be traced as external trace records.

Formats of Internal Trace Entries

Internal trace entries can be generated by applications from the GTRACE macro and by the GCS supervisor.

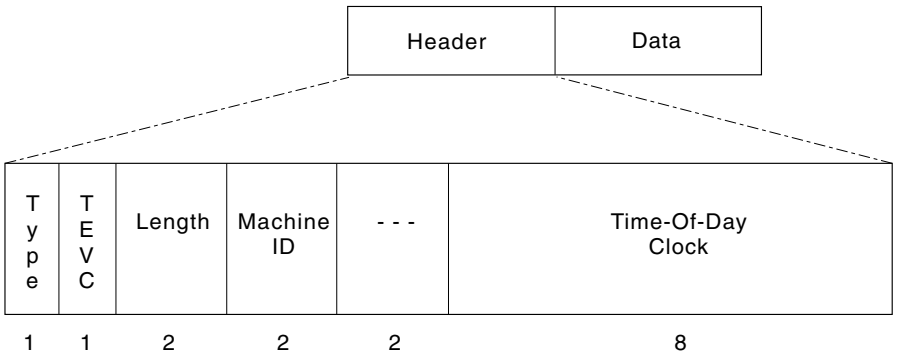
GCS trace entries consist of a common 16-byte header followed by event-specific data of up to 264 bytes.



Note: In the following diagrams, reserved fields are indicated by the word 'Reserved' or by dashes (- -).

Trace Header Format

The 16-byte header looks like this:



Type
shows the type of trace entry:

Hex Code	Trace Entry Type
01	Dispatcher
02	External interrupt
03	I/O interrupt
04	Program interrupt
05	SVC interrupt
06	I/O request
07	IUCV signal system service details
08	SVC GETMAIN request
09	SVC FREEMAIN request
0A	GETMAIN request through a branch entry
0B	FREEMAIN request through a branch entry
0C	APPC/VM synchronous event entry
0E	GTRACE macro data.

TEVC (trace entry verification code)
keeps track of every time the table *wraps around*. The first set of entries will have a TEVC of X'00'. Each time the table wraps around, this number increases by 1 until it reaches X'FF'. After that, it recycles to X'00'.

By looking at this number, you will be able to identify entries left over from the previous wraparound. This could be important, for example, if the GCS supervisor secures a trace table slot and then gets interrupted by CP before storing a new entry there. That slot would remain reserved, but unused, by the interrupted machine. Other machines in the group, when dispatched by CP, would create trace table entries in slots following it.

Length
contains the length of the whole entry, including this header. This length does not include the space that follows GTRACE entries which aligns the next trace table entry on a 32-byte boundary.

Machine ID
identifies the virtual machine associated with this entry. When the trace table is

located in common storage, there is a single trace table for the entire GCS group. It is important that you have the proper virtual machine identification.

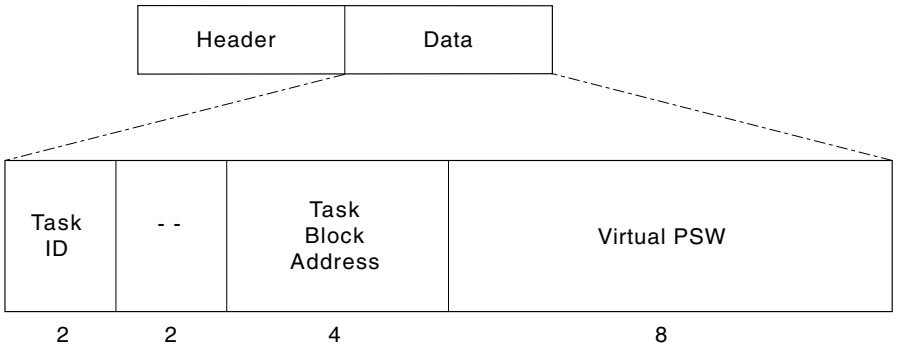
Time-Of-Day Clock
indicates what time this entry was created in time-of-day format.

Trace Data Format

The data portion of trace entries can have any of the following formats:

- Dispatcher (type X'01'), see page 107
- External interrupt (type X'02'), see page 108
- I/O interrupt (type X'03'), see page 110
- Program interrupt (type X'04'), see page 111
- SVC interrupt (type X'05'), see page 112
- SIO (type X'06'), see page 113
- IUCV signal system service (type X'07'), see page 114
- GETMAIN through an SVC (type X'08'), see page 115
- FREEMAIN through an SVC (type X'09'), see page 116
- Branch entry GETMAIN (type X'0A'), see page 117
- Branch entry FREEMAIN (type X'0B'), see page 118
- APPC/VM synchronous event (type X'0C'), see page 119
- GTRACE (typeX'0E'), see page120

Dispatcher (type X'01')

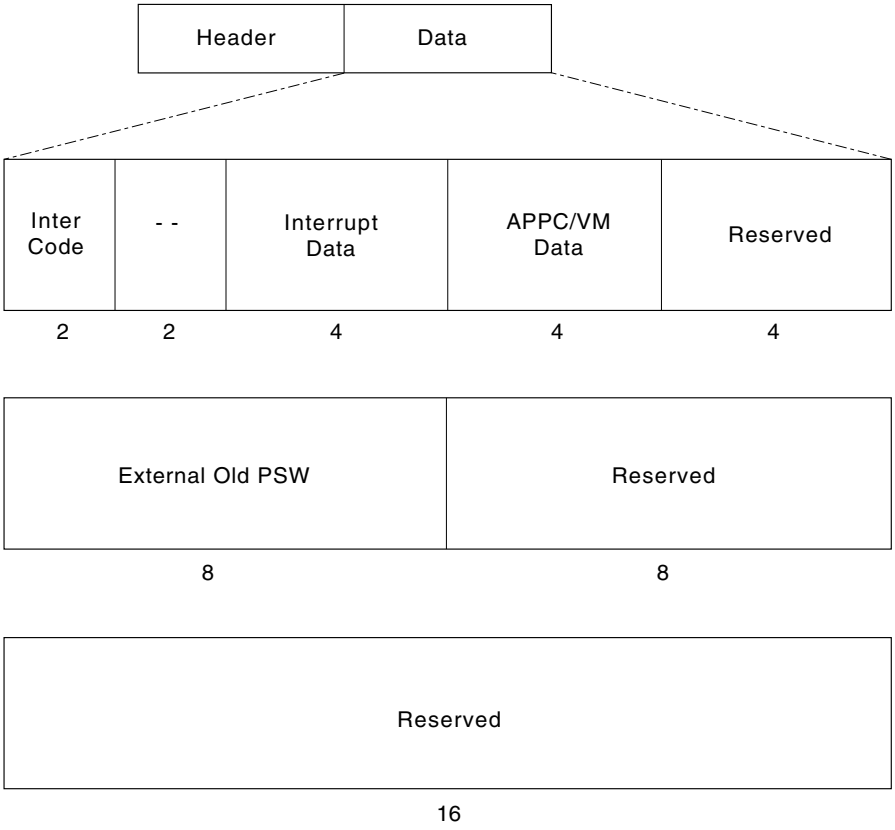


Task ID
identifies the task being traced.

Task Block Address
contains the address of a task control block for the task being dispatched.

Virtual PSW
contains the virtual PSW being dispatched.

External Interrupt (type X'02')



Inter Code
contains the External Interruption Code.

Interrupt Data
contains a value that depends on the type of external interrupt.

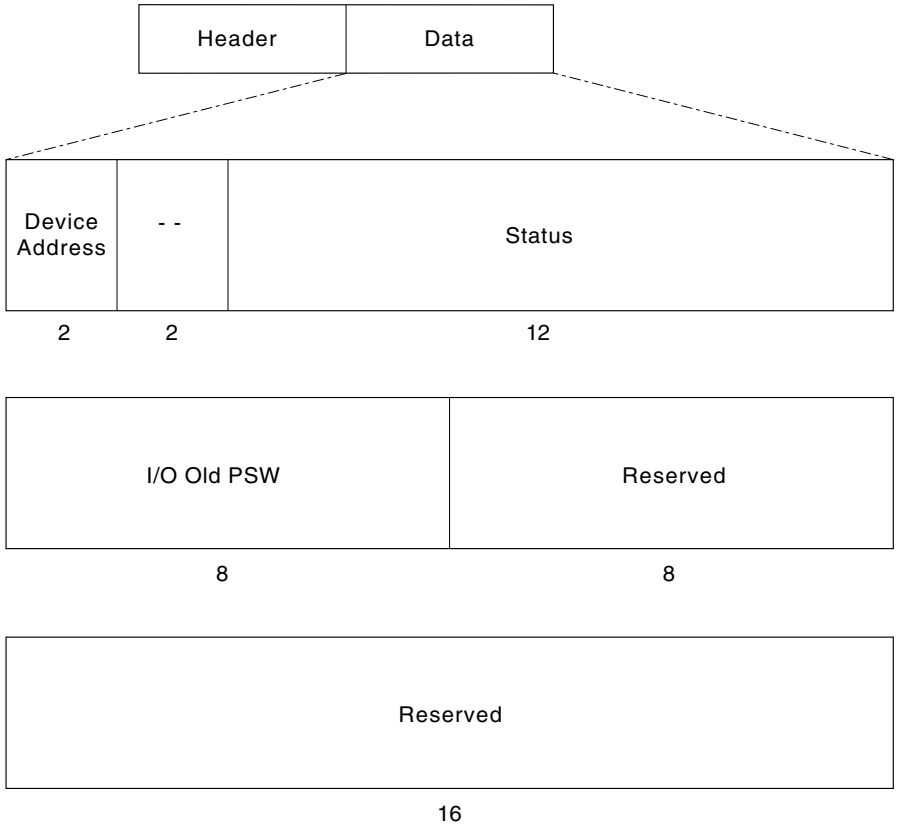
- For a timer interrupt (code X'1004') it contains a pointer to the timer queue element.
- For an IUCV or APPC/VM interrupt (code X'4000') it contains a:
  - 2-byte IPPATHID
  - 1-byte IPFLAGS1
  - 1-byte IPTYPE.
- For all other types of external interrupts this is a reserved field.

APPC/VM Data
contains APPC/VM data.

- For an APPC/VM interrupt (code X'4000' with an IPTYPE of X'81', X'82', X'83', X'87', X'88', or X'89'), it contains a:
  - 2-byte IPCODE.
  - 1-byte IPWHATRC—for a connect pending (type X'81') interrupt, this byte contains the IPFLAGS2 field.
  - 1-byte IPSENDOP.
- For all other types of external interrupts this is a reserved field.

External Old PSW
contains the external old PSW. If an IUCV poll (rather than an external interrupt) generates this entry, the external old PSW contains zeros (except for the interrupt code).

I/O Interrupt (type X'03')

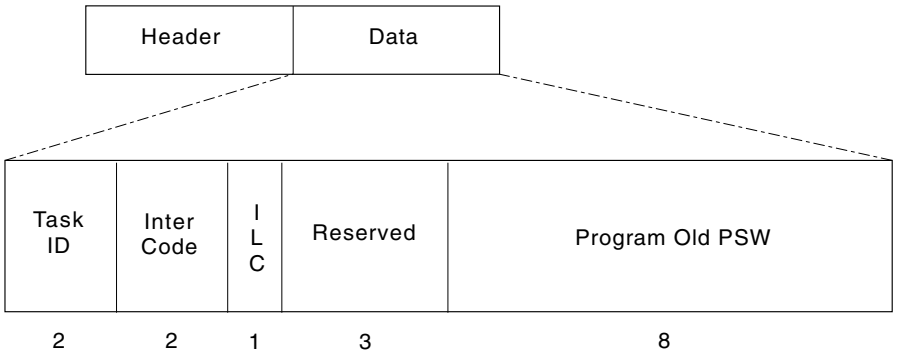


Device Address
contains the device number (2 bytes) of the interrupting device.

Status
contains the subchannel status word (SCSW, 12 bytes).

I/O Old PSW
contains the I/O old PSW (8 bytes).

Program Interrupt (type X'04')



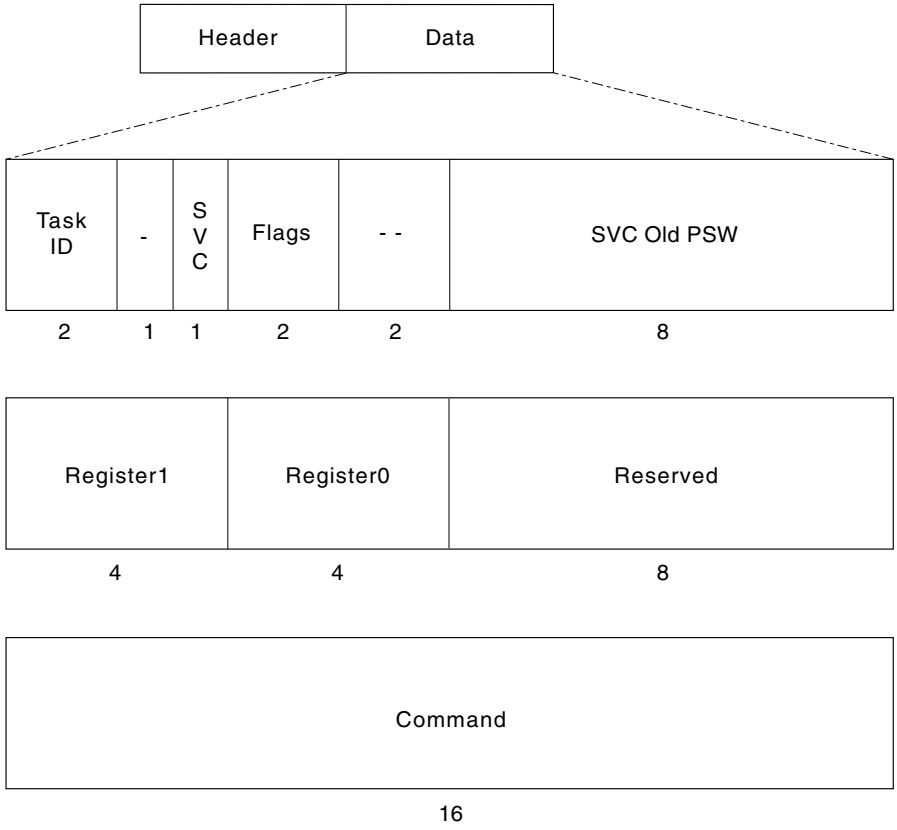
Task ID
identifies the task being traced.

Inter Code
contains the Program Interruption Code.

ILC
contains the Instruction Length Code.

Program Old PSW
contains the program old PSW.

SVC Interrupt (type X'05')



Task ID
identifies the task being traced.

SVC
is the number of the SVC entered by the invoker (1 byte).

Flags
is a reserved field for all but two SVCs.

For SVC 203, it contains the flag and code parameter.

For a DOS SVC, the leftmost bit of this field is set to one, and the rest of the 2 bytes is reserved.

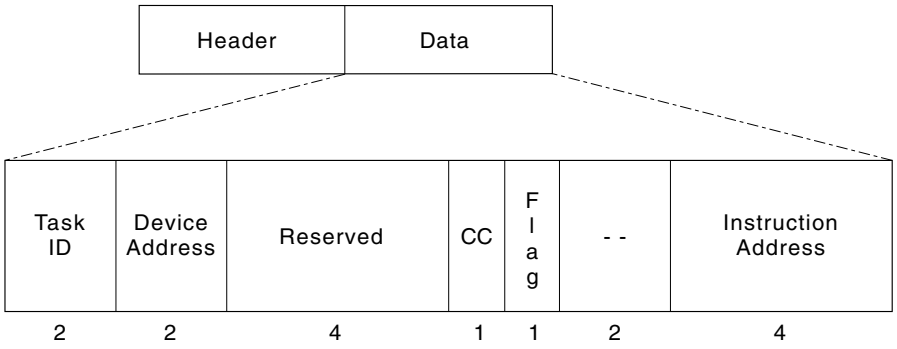
SVC Old PSW
contains the SVC old PSW (8 bytes) for all SVCs.

Register 1
contains the contents of register 1 for all SVCs.

Register 0
contains the contents of register 0 for all SVCs.

Command
contains the first 16 bytes of the command for an SVC 202.

SIO (type X'06')



Task ID
identifies the task being traced.

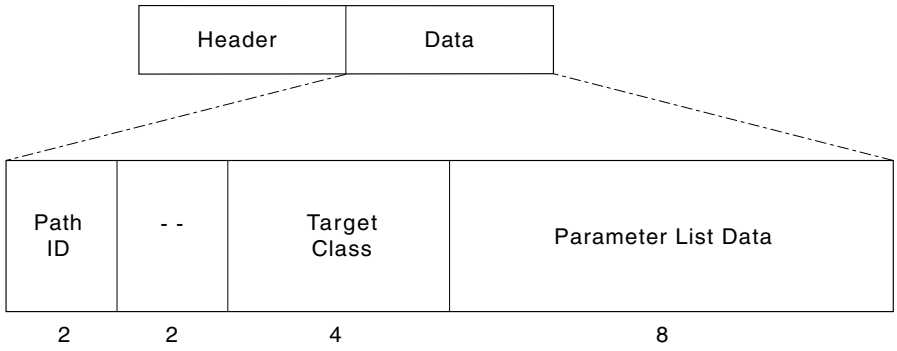
Device Address
contains the virtual address of the device to which a Start Subchannel (SSCH), Test Subchannel (TSCH), or Halt Subchannel (HSCH) command has been issued. For TSCH, this is the virtual channel address.

CC
contains the condition code from Start Subchannel operation. For GENIO START, it contains the condition code returned by the SSCH instruction. For GENIO STARTR, it contains the condition code returned by the DIAGNOSE code X'98' SSCH subfunction.

Flag
indicates a GENIO START or START function has been issued and that the CC field contains a valid condition code.

Instruction Address
contains the address of an I/O instruction or a DIAGNOSE.

IUCV Signal System Service (type X'07')

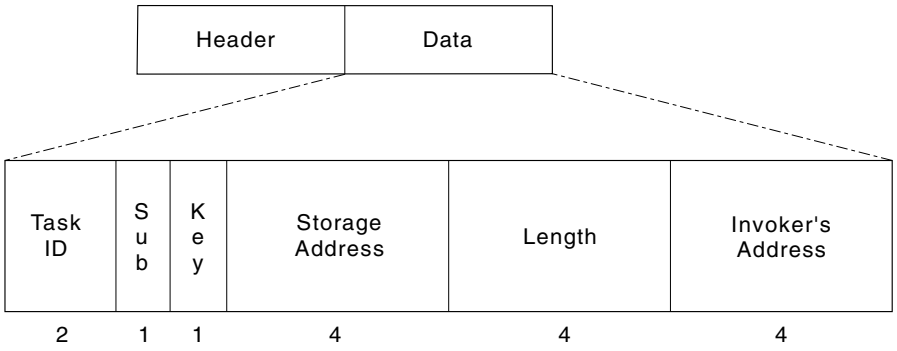


Path ID
identifies a 2-byte IUCV path.

Target Class
identifies an IUCV target class containing the interrupt source's signal ID and type of signal sent.

Parameter List Data
contains IUCV parameter list data.

GETMAIN via SVC (type X'08')



Task ID
identifies the task being traced.

Sub
identifies the subpool of storage being requested. It contains zeros when:

- An SVC 4 fails because of an incorrect parameter list address
- The GETMAIN fails because of an incorrect mode byte
- The requested subpool was zero

Key
contains the following information:

Bits	Description
0-1	contains LOC or position in storage where: **01** is below the line. **10** is resident storage. **11** is above the line.
2	Unused
3-6	contains the key of storage being obtained. It contains zeros when: • An SVC 4 fails because of an incorrect parameter list address. • The GETMAIN fails because of an incorrect mode byte. • If either the length or the subpool is incorrect.
7	contains the fetch-protection signal. The rightmost bit of this field serves as a fetch-protection signal. If the subpool of storage you request is **not** fetch-protected, this bit is 0 (zero).

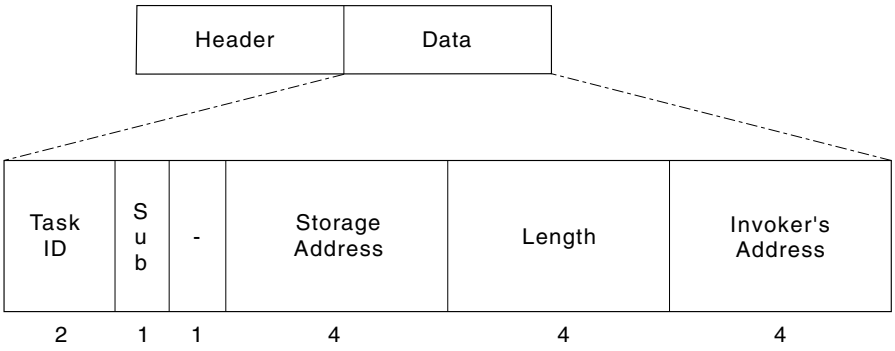
Storage Address
contains the address of storage obtained. If the GETMAIN failed, it contains zeros.

Length
contains the length of the storage requested. It contains zeros when:

- An SVC 4 fails because of an incorrect parameter list address.
- The GETMAIN fails because of an incorrect mode byte.

Invoker's Address
contains the address that follows the invoker's SVC.

FREEMAIN via SVC (type X'09')



Task ID
identifies the task being traced.

Sub
identifies the subpool of storage being released. If the FREEMAIN fails, it contains the subpool associated with the FREEMAIN.

It contains zeros when:

- An SVC 5 is entered with an incorrect parameter list address
- An unsupported MVS parameter is specified on the FREEMAIN macro
- An incorrect mode byte is encountered
- The requested subpool was zero.

Storage Address
contains the address of storage being released. If the FREEMAIN fails, it contains the storage address passed to FREEMAIN.

It contains zeros for the following failures:

- An SVC 5 is entered with an incorrect parameter list address
- An unsupported MVS parameter is specified on the FREEMAIN macro
- An incorrect mode byte is encountered.

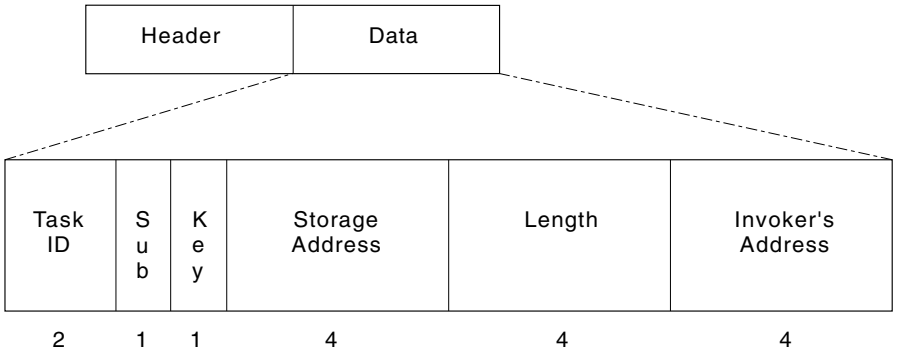
Length
contains the length of the storage released. If the FREEMAIN fails, it contains the length passed to FREEMAIN.

It contains zeros for the following failures:

- An SVC 5 is entered with an incorrect parameter list address
- An unsupported MVS parameter is specified on the FREEMAIN macro
- An incorrect mode byte is encountered.

Invoker's Address
contains the address that follows the invoker's SVC.

Branch Entry GETMAIN (type X'0A')



Task ID
identifies the task being traced.

Sub
identifies the subpool specified in the GETMAIN request.

Key
contains the following information:

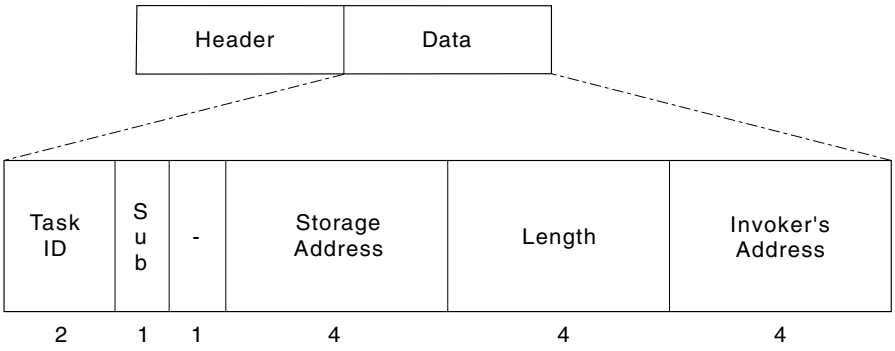
Bits	Description
0-1	contains LOC or position in storage where: **01** is below the line. **10** is resident storage. **11** is above the line.
2	Unused
3-6	contains the key of storage being obtained. It contains zeros when: • An SVC 4 fails because of an incorrect parameter list address. • The GETMAIN fails because of an incorrect mode byte. • If either the length or the subpool is incorrect.
7	contains the fetch-protection signal. The rightmost bit of this field serves as a fetch-protection signal. If the subpool of storage you request is **not** fetch-protected, this bit is 0 (zero).

Storage Address
contains the address of storage obtained. If the GETMAIN failed, it contains zeros.

Length
contains the length of the storage requested.

Invoker's Address
contains the address following the invoker's GETMAIN call.

Branch Entry FREEMAIN (type X'0B')



Task ID
identifies the task being traced.

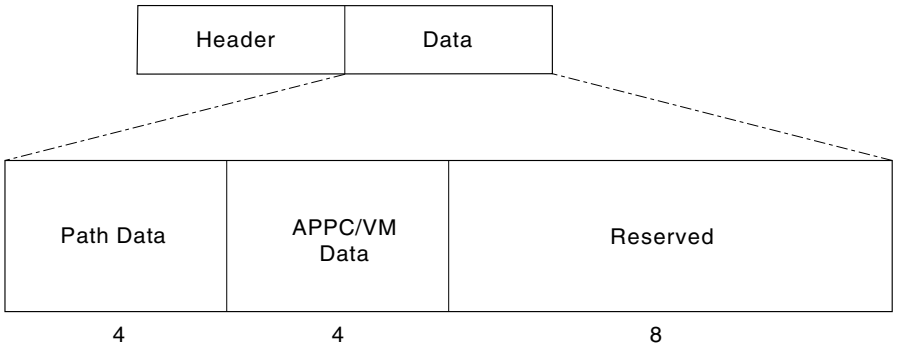
Sub
identifies the subpool specified in the FREEMAIN request.

Storage Address
contains the address of storage being released. If the FREEMAIN fails, it contains the storage address passed to FREEMAIN.

Length
contains the length of the storage released. If the FREEMAIN fails, it contains the length passed to FREEMAIN.

Invoker's Address
contains the address following the invoker's FREEMAIN call.

APPC/VM Synchronous Event (type X'0C')



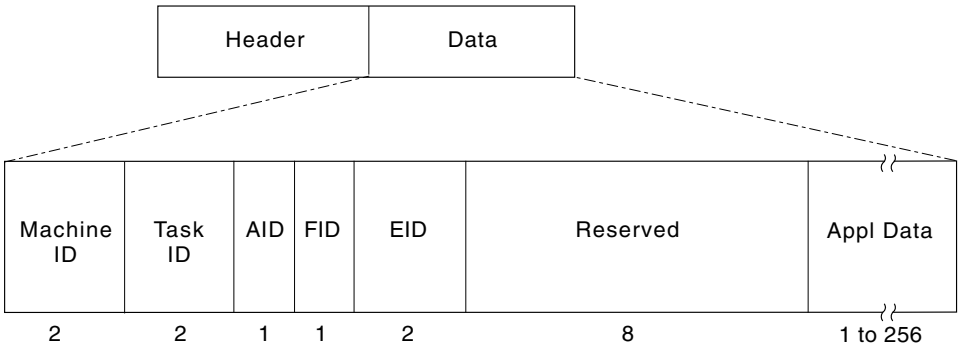
Path Data
contains a:

- 2-byte IPPATHID
- 1-byte IPFLAGS1
- 1-byte IPTYPE.

APPC/VM Data
contains a:

- 2-byte IPCODE
- 1-byte IPWHATRC–For a connect pending (type X'81') interrupt, this byte contains the IPFLAGS2 field
- 1-byte IPSENDOP.

GTRACE (type X'0E')



Machine ID
identifies the virtual machine associated with this entry. When the trace table is located in common storage, there is a single trace table for the entire GCS group. It is important that you have the proper virtual machine identification. This ID is the same data that is in the header.

Task ID
identifies the task being traced.

AID
indicates this is a data record. It always contains X'FF'.

FID (format ID)
identifies the formatting module that handles this entry.

EID
contains information from the GTRACE macro's ID parameter.

Appl Data
is up to 8192 bytes of data provided by the application. If internal tracing is being used, then the maximum is 256 bytes.

Service Point (SP) Trace Entries

The SP trace entries appear as GTRACE records in the trace table with the following EIDs and application data:

X'E400'	Branch Entry to WAIT	Data=	Register1 (4 bytes) Register14 (4 bytes)
X'E401'	Branch Entry to SCHEDEX	Data=	The SCHEDEX Data Area (85 bytes)

X'E402'	Branch Entry to IUCVCOM	Data=	The IUCVCOM parameter list (40 bytes)
X'E404'	Branch Entry to VALIDATE	Data=	Register0 (4 bytes) Register1 (4 bytes) Register2 (4 bytes) Register14 (4 bytes)
X'E405'	Branch Entry to POST	Data=	Register1 (4 bytes) Register14 (4 bytes)

See the ITRACE command in *z/VM: Group Control System* for more information on SP trace entries.

External Tracing Facilities

You can collect trace data in the system trace files for later formatting and viewing. This requires entering the following commands:

1. The TRSOURCE and TRSAVE commands
2. The ETRACE command.

See the *z/VM: CP Commands and Utilities Reference* for more information on the TRSOURCE and TRSAVE commands; see the *z/VM: Group Control System* book for more information on the ETRACE command.

The users who enter the TRSOURCE command must have a Class C privilege user ID. After the TRSOURCE commands have been entered, this machine can enter the ETRACE command to commence tracing for its own application or ETRACE GROUP for tracing the entire group (if it's an authorized machine). Use ETRACE to specify which of the following events should be traced and recorded in the spool file:

- Task dispatches
- External interrupts
- I/O interrupts
- Program interrupts
- SVC interrupts
- I/O requests (SIO and Diagnose)

- IUCV signal system service details
- APPC/VM synchronous events
- GETMAIN requests
- FREEMAIN requests
- User trace data generated using the GTRACE macro.

The CP TRACERED utility provides the facility with processing of trace data defined by the TRSOURCE command in system trace files. TRSOURCE defines what is going to be traced. TRSAVE specifies where data from traces defined by the TRSOURCE command are to be saved. CP and virtual machine data can be merged to produce a consolidated output file in chronological sequence.

See “I/O Debugging” on page 169 for guidance information on setting up tracing activity for CP, GCS, VSCS, and VTAM.

Using the TRSOURCE Command

TRSOURCE controls and displays CP data and I/O tracing activity. It supports definitions and control of I/O, data, and guest tracing. It controls and displays the status of guest tracing.

When you use the BLOCK option of TRSOURCE, trace data to be recorded is buffered by GCS. In this mode, if the GCS supervisor fails or a system reset occurs, the data remaining in the trace buffer cannot be sent to CP and is not recorded by CP. However, if a dump is available, you can view the trace data remaining in the buffer by finding the pointer to it in a system control block. See “Locating the External Trace Buffer.”

When you use the EVENT option of TRSOURCE, each trace record is sent directly to CP with no buffering of the trace data. When the trace is running in EVENT mode, there is no loss of data in the event the virtual machine loses control from CP, but the performance gain of BLOCK mode is lost.

Locating the External Trace Buffer

You can locate the external trace buffer by doing the following:

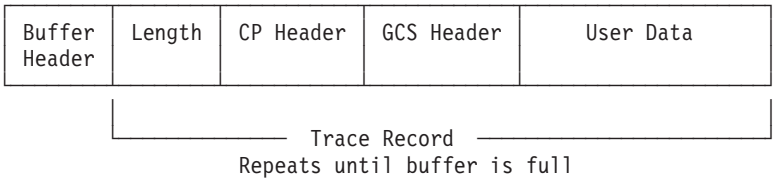
Locate the SI extension (SIE) address in the NUCON at displacement X'5C4'

Locate the TAB address in the SIE at displacement X'A0'

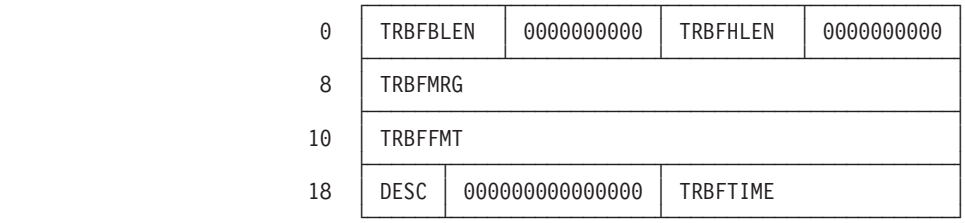
Locate the external trace buffer address in the TAB at displacement X'C'.

Format of the External Trace Buffer

The format of the external trace buffer is:



The layout of the buffer header is:

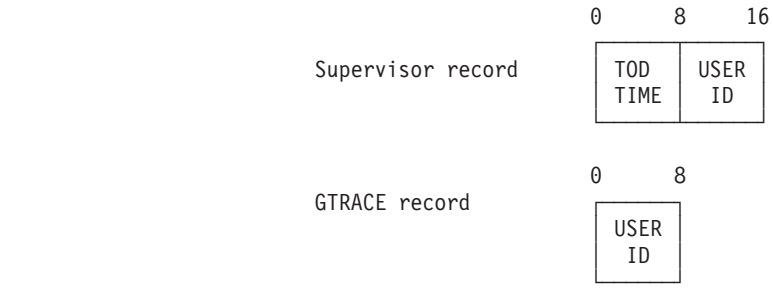


Disp	Name	Length (bytes)	Description
0	TRBFBLEN	2	Number of bytes filled in buffer
2	TRBFRES1	2	Reserved
4	TRBFHLEN	2	Length of buffer header W/O TRBFBLEN and TRBFRES1
6	TRBFRES2	2	Reserved
8	TRBFMRG	8	Name of merge routine
10	TRBFFMT	8	Name of format routine
18	TRBFDESC	1	Block descriptor set by CP on call to DIAGNOSE X'E0'
19	TRBFRES3	3	Reserved
1C	TRBFTIME	4	Time zone differential

The layout of the CP header is:

Disp	Name	Length (bytes)	Description
0	CPHTYPE	1	The type of CP header (3D)
1	CPHHLEN	1	The length of the CP header (in doublewords) (Headers must end on doubleword boundary)
2	CPHCODE	2	Individualizing code
2	CPHRESER	1	Reserved byte
3	CPHINDIV	1	GCS individualizing code byte (The individualizing code values are the same as the values in the TYPE field in HEADER1 of the internal trace records.) See "Formats of Internal Trace Entries" on page 104.
4	CPHLTH	2	Length of header + length of data (This length includes the CPH DSECT and the user data. It does not include any round up of the record to a 16 byte (DWORD) boundary.)
		2	Reserved

The layout of the GCS header is:



The layout of the user data is the same as for internal tracing entries. (See “Internal Tracing Facilities” on page 103).

Using the TRSAVE Command

TRSAVE specifies where data is to be saved. Data from traces defined by TRSOURCE may be saved in a system trace file.

A TRSOURCE/TRSAVE Command Example

The following is an example of a guest trace invoked by using the TRSOURCE and TRSAVE commands:

```
trsource id g1 type gt for rscs
trsave id g1 size 40 keep 3
trsource enable id g1
trsource disable id g1
trsource drop id g1
```

Note: Between the commands TRSOURCE ENABLE ID G1 and TRSOURCE DISABLE ID G1 in the above example, all tracing from the RSCS virtual machine is collected in a TRFILE with file name G1.

Using the CP TRACERED Utility

The CP TRACERED utility reads and formats trace data. All files to be processed must have been created under a valid current release. A total of five system trace files can be merged. Only one CP system trace table file or tape may be included. Therefore, you may specify one of the following:

- One CP system trace table file with up to four TRSOURCE trace files
- One CP tape with up to four TRSOURCE trace files
- Up to five TRSOURCE trace files.

A TRACERED Utility Example

The following is an example of CP data merged with virtual machine data:

If you entered the following:

```
tracered 0003 0004 cms cpvm out a (all hex
```

you might receive the following output:

```
----------------------------   04/06/95 16:54:06   ----------------------------

CPU   TOD CLOCK    CODE  ***********  TRACE ENTRY CONTENTS **********


----------------------------   04/06/95 16:54:32   ----------------------------

     33D17D82A640        A3888540 8481A381 40A396C2 C540D9C5 C3D6D9C4 SPID 0003
                         C5C4C9E2 40F3F240 E4E2C5D9

----------------------------   04/06/95 16:56:00   ----------------------------

     33D452F96400        A3888540 8481A381 40A396C2 C540D9C5 C3D6D9C4 SPID 0003
                         C5C4C9E2 40F3F240 E4E2C5D9

----------------------------   04/06/95 16:56:19   ----------------------------

0000 33E68389D640  0600  4C4C4C4C 00000026 008B1008 00FAD000 80522988 SPID 0004
0000 33E6838C9480  3300  D2C34040 004D3720 00FAD000 00FFD580 805229C8 SPID 0004
0000 33E6838DA260  2C00  00522178 00E5E2D7 00FFB180 8050570E 8052219C SPID 0004
0000 33E6838EA840  0700  4C4C4C4C 00000001 0086E008 00FAD000 8050572E SPID 0004
0000 33E683913620  2C00  00000004 00D8C3D8 00FFDE80 805036BC 805055A0 SPID 0004
0000 33E683926000  2200  0086F008 80001010 00FAD000 00FAD000 0031FCB8 SPID 0004
```

Using the QUERY TRFILES Command

Use the QUERY TRFILES command to display information about system trace files that you own. This includes the spool ID, file name, and time of creation.

General Trace Information

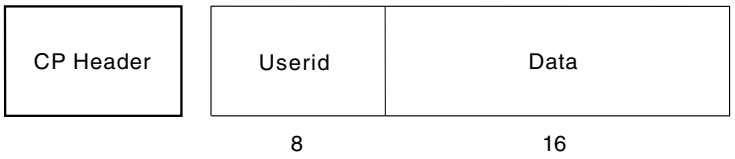
You can find general information about external tracing in the *z/VM: Group Control System* book.

Formatting and Displaying External Trace Records

The external trace file contains two different entries produced by GCS virtual machines for:

- GCS supervisor records
- GTRACE records.

The format for supervisor records is as follows:

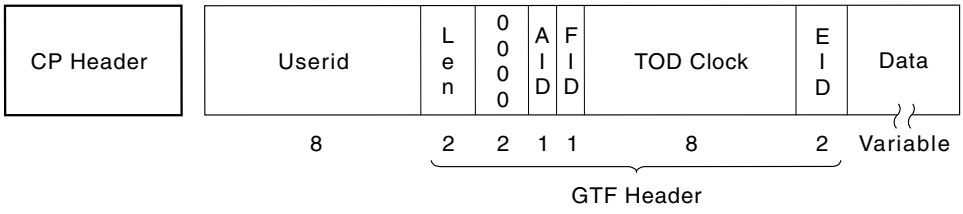


CP Header
contains an 8-byte header appended by CP when it gets the record.

User ID
identifies the virtual machine that the entry belongs to.

Data
contains the data portion of the event's internal trace entry. [4]

The format for GTRACE records is as follows:



CP Header
contains an 8-byte header appended by CP when it gets the record.

User ID
identifies the virtual machine that the entry belongs to.

Len
contains the length of the entry, including the GTF header.

0000
is a reserved field in the GTF header.

4. Internal trace entry formats are described in "Formats of Internal Trace Entries" on page 104.

AID
contains X'FF', indicating that this is a data record.

FID (Format ID)
identifies the formatting module used for this record.

TOD Clock
indicates when the record was built, in time-of-day format.

EID
contains information from the GTRACE macro's ID parameter.

Data
contains the internal trace entry without the internal header (up to 8k).

The main reason you create an external spool file with TRSOURCE is to print out or interactively display your trace information. The TRACERED utility [5] lets you do that by formatting trace entries in your external spool file and then printing your external file or creating a CMS file. The TRACERED utility handles the formatting of supervisor and GTRACE entries, sending both to a common format routine (GCTYTD). The TRACERED utility formats supervisor records and GTF header information. However, the applications being traced by means of the GTRACE facility have to supply their own GTRACE formatting modules. If they do not, their trace entries for the data portions of the records are printed unformatted, in hexadecimal.

As TRACERED goes through the spool file, it examines each entry one by one. Trace entries, which were recorded by GCS using a "MC 1, 10" instruction, are passed to a GCS module GCTYTD for formatting.

For supervisor records, GCTYTD calls a GCS-supplied formatting routine named GCTYTS to format it. However, for GTRACE records, GCTYTD uses GCS-supplied formatting routines to format the GTF header part of the record. GCTYTD also looks for another formatting routine, one supplied by the traced application, to finish the data portion of the record. (It uses the GTRACE record's 1-byte FID field to locate this routine. The routine's name must be GCTYTX*xx*, with *xx* being the 2-digit FID, and it must have a file type of TEXT.)

If the GCTYTD program cannot find a user-supplied formatting routine, it prints the entry information in hexadecimal. If the program does find a GCTYTX*xx* TEXT, it calls that routine.

For information about coding user-supplied formatting routines, including register contents at the time they are called by the GCTYTD program, see the GTRACE Macro in the *z/VM: Group Control System*.

Examples of Formatted External Trace Table Entries

Here are several sample supervisor event entries as you would see them in your external trace file.

- An entry type X'03' for an I/O interrupt:

5. TRACERED is a CP data reduction utility that works on the system trace file created by TRSOURCE. For more information on TRACERED, see the *z/VM: CP Commands and Utilities Reference* book.

```
3D 03  useridxx  VM/GCS I/O INTERRUPT
        DEVICE ADDRESS = xxxx
        STATUS = xxxxxxxx x x xxxxxx xx xx xxxx
        OLD PSW = xx x x x x xx xxxxxxxx
```

- An entry type X'05' for an SVC interrupt:

```
3D 05 useridxx   VM/GCS SUPERVISOR CALL INTERRUPT
        SVC CODE = xx
        TASK ID = xxxx
        FUNCTION NAME = xxxxxxxx
        PARM BYTES 8-15 = X'xxxxxxxxxxxxxxxx' = *xxxxxxxx*
        REGISTER 1 = xxxxxxxx
        REGISTER 0 = xxxxxxxx
        OLD PSW = xx x x x x  xx xxxxxxxx
```

- An entry type X'08' External Trace Table Entry by SVC GETMAIN:

```
3D 08 useridxx   VM/GCS GETMAIN VIA SVC
       TASK ID = xxxx
       KEY = x   xxxxxxxxxxxxxxxxxx
       SUBPOOL = xx
       STORAGE ADDRESS = xxxxxxxx
       LENGTH = xxxxxxxx
       ISSUER ADDRESS = xxxxxxxx
       LOC = xxxxx
```

- An entry type X'09' External Trace Table Entry by SVC FREEMAIN:

```
3D 09 useridxx   VM/GCS FREEMAIN VIA SVC
       TASK ID = xxxx
       SUBPOOL = xx
       STORAGE ADDRESS = xxxxxxxx
       LENGTH = xxxxxxxx
       ISSUER ADDRESS = xxxxxxxx
```

- An entry type X'0A' External Trace Entry by Branch Entry GETMAIN:

```
3D 0A useridxx   VM/GCS GETMAIN VIA BRANCH ENTRY
       TASK ID = xxxx
       KEY = x   xxxxxxxxxxxxxxxxxx
       SUBPOOL = xx
       STORAGE ADDRESS = xxxxxxxx
       LENGTH = xxxxxxxx
       ISSUER ADDRESS = xxxxxxxx
       LOC = xxxxx
```

- An entry type X'0B' External Trace Entry by Branch Entry FREEMAIN:

```
3D 0B useridxx   VM/GCS FREEMAIN VIA BRANCH ENTRY
       TASK ID = xxxx
       SUBPOOL = xx
       STORAGE ADDRESS = xxxxxxxx
       LENGTH = xxxxxxxx
       ISSUER ADDRESS = xxxxxxxx
```

- An entry type X'0E' for a GTRACE entry:

```
3D 0E useridxx   VM/GCS USER REQUESTED GTRACE
        TIME OF DAY CLOCK = xxxxxxxxxxxxxxx
        LENGTH OF GTF HEADER AND TRACE DATA = xxxx
        FORMAT ROUTINE ID = xx
        EVENT IDENTIFICATION = xxxx
       [formatted GTRACE data appears here. . . .]
```

Dumping Facilities

The Common Dump Receiver

To let you dump out the contents of virtual storage and see where problems have occurred, GCS must provide a way around its own safeguard mechanisms. Otherwise, your GCS dumps would be largely incomplete.

Rules of Authorization

If a dump is directed to an authorized user, all of the requested storage is dumped, including the saved segments. If the dump is directed to an unauthorized user, only the storage with a key of 14 and nonfetch-protected storage is to be dumped.

If you direct the dump to yourself or to another unauthorized user ID, you cannot dump any fetch-protected areas or storage with a key other than 14. Unauthorized dump receivers can accept only key-14 and other nonfetch-protected storage.

You can solve this problem by singling out one authorized virtual machine as your common dump receiver. At build time, when creating your GCS configuration file, you are prompted to name this common dump receiver. Choose any authorized user ID, perhaps the same user ID that you specify as your recovery machine. Be sure you list it on the GROUP EXEC's screen of authorized GCS user IDs. If you name a common dump receiver, GCS's dump functions, described in "Creating GCS Dumps" on page 130, automatically send their output to it. [6]

Interactive Debugging Support

Using Authorized Control Program (CP) Commands

Authorized user IDs can have access to the following CP debugging commands:

- BEGIN
- DISPLAY
- STORE
- DUMP

Initially, these are Class G commands, available to all user IDs. You may want to reclassify these commands to prevent unauthorized users from altering storage that may effect other members of the GCS group.

For more information on controlling access to CP commands, see the *z/VM: CP Planning and Administration* book.

6. The exception is GDUMP, which optionally lets you choose another receiver.

Analyzing Dumps

After storage has been dumped, it can be:

- Read into a CMS file
- Analyzed by the receiving virtual machine under CMS with the Dump Viewing Facility
- Dumped to tape (using spool-to-tape) and sent to an IBM support center for analysis.

The Dump Viewing Facility uses specialized routines for formatting GCS dumps.

To use the Dump Viewing Facility successfully in processing a virtual machine dump, the minidisks containing GCS must be accessed before processing the dump by the Dump Viewing Facility.

Dump Viewing Facility Features for GCS Dumps

Creating a GCS module map —
the Dump Viewing Facility MAP command looks for a nucleus load map with the default name GCSNUC MAP *. It creates a module map called GCSDVF MAP that contains a header and a compressed version of the load map. The Dump Viewing Facility ADDMAP command appends the module map to a formatted dump.

Printing formatted VTAM or VSCS control blocks —
Use the Dump Viewing Facility PRTDUMP command to specify whether you want formatted VTAM or VSCS control blocks printed in a dump. You'll have this option for any GCS- or VMDUMP-generated dump of type GCS or RSCSV2. First you receive a prompt asking you if you want your dump printed using the VTAM option. If you do not pick the VTAM option, you receive a prompt asking you if you want your dump printed using the VSCS option. If you do not choose either option, only summary information from the dump is printed.

Viewing the GCS dump information —
Use the Dump Viewing Facility DUMPSCAN command to format a GCS dump and to view it, along with the appended module map, interactively. Use the Dump Viewing Facility BLOCKDEF utility to format control blocks and to view them interactively. For more information, see “Processing GCS Dumps with the Dump Viewing Facility” on page 137 and the *z/VM: Dump Viewing Facility* book.

Using the GCS debug tools —
Additional GCS debugging tools are available and may be helpful when diagnosing GCS problems. When run against an existing GCS dump, these tools may help by formatting trace tables, calculating storage used by load modules, as well as mapping storage used by particular tasks. Other useful debugging capabilities are also included with this tools package.

The GCSDUMP tools package is shipped with z/VM version 5 release 2 on an “as is” basis, optionally installed on the MAINT 193 disk. For more information, see the README SAMPGCS file that comes with package on MAINT's 193 disk.

Dumping VSAM Information

When VSAM detects certain internal logic errors, it produces a special dump, called an IDUMP, that can help you identify those problems. To look at information in the

dump header, use the DUMPID subcommand of DUMPSCAN. This dump header contains the following information:

VSAM IDUMP	24–character symptom string	MM/DD/YY	HH:MM:SS	SAVEAREA ADDR

VSAM IDUMP
is a dump identification message.

24-character symptom string
identifies error codes, the location of the error, and the module that detected the error. For information on how to interpret this character string, see *VSE/VSAM Programmer's Reference*.

MM/DD/YY
is the date when VSAM detected the error.

HH:MM:SS
is the time of day when VSAM detected the error.

SAVEAREA ADDR
contains the address of the save area that shows what each register contained when VSAM discovered the error. Ignore the first 16 bytes of this save area, and look for the register contents beginning at the 17th byte. You will find the contents of all 16 registers in the following order: registers 9–15, registers 0–8.

Creating GCS Dumps

GCS uses the CP VMDUMP command to produce dumps of virtual machines. Dumps are always sent to a virtual reader.

Dumps are produced several ways:

- By using the GDUMP command
- From an application using the SDUMP, SDUMPX macro or ABEND macro
- By entering the CP command, SYSTEM RESTART
- After an abnormal end of the GCS supervisor or an abnormal termination of a program.

  All GCS dumps follow the same rules for authorization. If the receiver of the dump is not authorized, he receives only key 14 and other non-fetch-protected storage. If the receiver is authorized, all areas of the virtual machine and any saved segments can be dumped.
- By using the VMDUMP command.

  If the receiver is authorized, all areas of the virtual machine and any saved segments can be dumped.

The GDUMP Command

GDUMP is a GCS command. When you enter the GDUMP command, a "snapshot" of the virtual machine's storage is taken. You can spool the dump to a common dump receiver's virtual reader, to a specified user's virtual reader, or to the issuer's virtual reader. You can dump specific ranges of storage by specifying it on the GDUMP command. For further information on GDUMP and dump authorization, see the *z/VM: Group Control System* book.

The SDUMP Macro

SDUMP is a macro that you can start during GCS processing. It takes a dump of the GCS system and continues processing. The resulting dump contains the storage of the issuer's virtual machine. SDUMP is spooled to a common dump receiver or to the issuer's virtual reader. All or portions of storage may be requested when using the SDUMP macro. A dump will not be taken if SET DUMP OFF has been issued. For further information on SDUMP, see the *z/VM: Group Control System* book.

The SDUMPX Macro

Use the SDUMPX macro when you are running an XC virtual machine and wish to dump part or all of a data space that you are accessing. For further information on SDUMPX, see the *z/VM: Group Control System* book.

The ABEND DUMP Macro

Conditions can occur within a program that may force an abnormal ending (abend) and cause the dumping of the system registers and storage. When this happens, an abend dump is produced. In addition to the "forced abend", a program may also choose to generate an abend condition by issuing its own ABEND MACRO. The dump contains the entire virtual machine as well as any discontiguous shared segments (shared segments linked to your GCS system, but not within the bounds of your virtual machine). GCS uses this facility just as CMS and CP do, except that the dump is spooled to the common dump receiver if one was specified at GCS build time (in the GROUP EXEC), rather than the user's virtual reader.

Note: The DUMP operand is overridden by the SET DUMP command. SET DUMP ON implies that the dump is always issued. SET DUMP OFF implies the dump is not issued. If you enter SET DUMP DEFAULT, the DUMP operand from ABEND takes preference.

For further information on abend dumps, see "Processing Abends" on page 135.

The SYSTEM RESTART Command

GCS has the capability to dump a virtual machine's storage and any saved segments when that virtual machine issues the CP command SYSTEM RESTART. This is helpful when you cannot use the GDUMP command, for example, if you have a GCS disabled loop and enter **#cp system restart**.

As with other GCS dumps, the resulting dump from the SYSTEM RESTART is in VMDUMP format and is spooled to a common dump receiver. If you don't have a common dump receiver, the data goes to the machine that issued it. A SYSTEM RESTART dump follows the same rules of authorization as other GCS dumps, when determining what storage to dump.

The VMDUMP Command

VMDUMP is issued in GCS in the same manner as in CMS. When you enter the VMDUMP command, a *snapshot* of the system is taken. This *snapshot* is then spooled to your virtual reader. For further information on the VMDUMP command, see the *z/VM: CP Commands and Utilities Reference*.

Preserving Common Storage

To produce a dump requested by one of these functions, GCS calls CP and requests a dump. While it performs the dump, CP continues dispatching other

machines in the virtual machine group. This poses a problem if those members go on to change common storage as it is being dumped.

To preserve common storage contents until the dump finishes, the GCS supervisor acquires the common storage lock. This prevents other machines from acquiring the lock during the dump. If all authorized machines test the common lock before trying to change common storage, they will be effectively suspended until the dump finishes. The only common storage that might change is that obtained by other machines before the dump began.

If the SET DUMPLOCK OFF command was entered, the common storage lock is not held while GCS is dumping. Other virtual machines running in the group can then alter common storage.

Note: The common storage lock gets set *on* only if your common dump receiver is an authorized GCS user ID and you are using the SDUMP and GDUMP functions. It is possible to receive two dumps. An example of this would be if a user ran out of storage while producing a dump. One dump would be produced as the user dump, and the second dump would be the supervisor dump.

How to Find the GCS Virtual Machine That Created a Dump

When you process a GCS dump by the DUMPLOAD utility, the user ID of the virtual machine where the dump was produced is not kept for use by the Dump Viewing Facility. Therefore, situations may arise where you have several dumps on the minidisks and you need to know which virtual machine has created them. Use the DUMPSCAN DISPLAY 204 subcommand to view the NUCVMID field. NUCVMID is an 8-byte field that contains the virtual machine user ID, as specified in the CP directory.

Using the GCS Trace Facilities

The GCS trace is a powerful tool used to help track down the source of a problem.

GCS has two tracing facilities:

- Internal trace (ITRACE)
- External trace (ETRACE).

These tracing facilities record events while GCS is running. You can specify which events to track on the ITRACE or ETRACE commands. For information on how to use these commands, see the *z/VM: Group Control System* book.

ITRACE

The internal trace facility records specific events as they occur in the GCS system. Internal tracing of GCS supervisor events is automatically enabled at IPL, but the user may disable this tracing if so desired. Unauthorized users can disable events for themselves only if these events have not been enabled for the entire group.

Locating the GCS Internal Trace Table

Because the internal trace table can be in either private or common storage, you need to determine in which storage the active trace table is located.

If you are in interactive mode, enter the QUERY TRACETAB command. The response you receive tells you in which storage location the trace table is now being maintained.

If you are working from a dump:

Use the load map to locate the address of the GCTGST.

Locate the group flag in the GST at displacement X'14'. If the flag contains:

X1XX XXXX	trace table is in private storage
X0XX XXXX	trace table is in common storage.

In Private Storage

You can locate the internal trace table in private storage by doing the following:

Locate the SI Extension (SIE) address in the NUCON at displacement X'5C4'

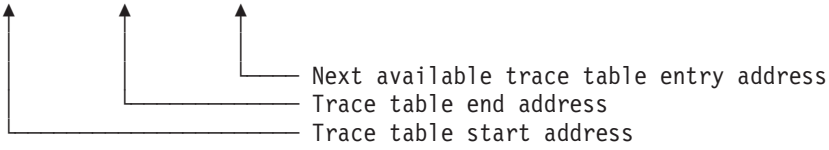
Locate the private trace anchor table (TAB) address in SIE at displacement X'A0'

The TAB contains the following pointers to the internal trace table when it is in private storage.

Displacement	Field Description
X'10'	The starting address of the trace table
X'14'	The ending address of the trace table
X'18'	The address of the next available trace table entry

Following is an example for locating the trace table in private storage:

```
cp d 5C4.4
R000005C4 00000750
Ready;
cp d 7F0.4
R000007F0 00002B28
Ready;
cp d 2B38.C
R00002B38 00FE0000 00FE4000 00FE2BC0
Ready;
```



In Common Storage

You can locate the internal trace table in common storage by doing the following:

Locate the common trace block (CTB) address in NUCON at displacement X'21C'

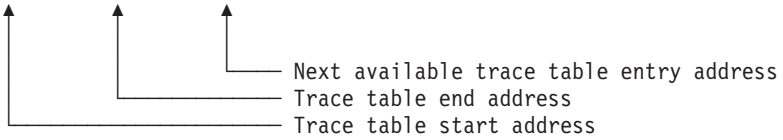
The CTB contains the following pointers to the internal trace table when it is in common storage.

Displacement	Field Description
X'00'	The starting address of the trace table
X'04'	The ending address of the trace table
X'08'	The address of the next available trace table entry

If these first three fields in the CTB are zero, tracing is being done in private storage.

Following is an example for locating the trace table in common storage:

```
cp d 21c.4
R0000021C 0083EB60
Ready;
cp d 83eb60.c
R0083EB60 0083F000 00843000 00841C60
Ready;
```



Locating the Last Trace Entry in Storage or in a Dump

To find the last trace entry, use the pointer to the next available trace entry. Trace entries created by the GCS supervisor are 32 (X'20') or 64 (X'40') bytes long. Trace entries created by the GTRACE macro have variable lengths (consisting of a fixed 32-byte area and 1 to 256 bytes of data). Trace entries which follow GTRACE entries are aligned on a 32-byte boundary, and the space between these entries is filled with binary zeros.

If the trace table does not contain any GTRACE entries, find the last entry by subtracting 32 (X'20') or 64 (X'40'), depending on the type of supervisor trace entry (see “Formats of Internal Trace Entries” on page 104), from the pointer to the next available trace entry. If the trace table contains GTRACE entries, you have to know the layout of those trace entries to be able to find the last trace entry.

Using the Trace Table

Each supervisor trace table entry is 32 or 64 bytes long. The first 16 bytes are the header. The header describes what type of event is being recorded, the time of the event, and for which virtual machine the event is being recorded. The remaining bytes contain information unique to the recorded trace event. Trace entries created by GTRACE macro have variable length. Trace entries which follow GTRACE entries are aligned on a 32-byte boundary, and the space between these entries is filled with binary zeros. For further information on trace table entries, see “Formats of Internal Trace Entries” on page 104.

To see which events were being traced for a virtual machine in a dump, look at the trace anchor block (TAB), as follows:

1. Locate the SI Extension (SIE) address in the NUCON at X'5C4'
2. Find the TAB address at SIE + X'A0'.

The TAB contains the following information:

Displacement	Field Description
X'00'	The address of the CTB
X'04'	Flags for external tracing

Byte	Field Description
1xxx xxxx	Dispatcher
x1xx xxxx	External interrupts
xx1x xxxx	I/O interrupts
xxx1 xxxx	Program interrupts

	xxxx 1xxx	SVC interrupts
	xxxx x1xx	I/O requests
	xxxx xx1x	Signal system service events
	xxxx xxx1	GTRACE events

X'05'

Byte	**Field Description**
1xxx xxxx	GETMAIN requests
x1xx xxxx	FREEMAIN requests
xx1x xxxx	APPC/VM synchronous events
xxx1 1111	Reserved

X'06' Flags for Internal Tracing

Byte	**Field Description**
1111 11xx	Reserved
xxxx xx1x	Supervisor events
xxxx xxx1	GTRACE events

When a tracing flag is on, that event is being traced for the subject virtual machine.

ETRACE

The external trace facility records specific events within a group as they occur in the GCS system. These events are recorded in one or more system trace files by the CP TRSOURCE command. This spool file may optionally be a wraparound file. In order to use the ETRACE facility, a user with VM privilege class C must first enter the CP TRSOURCE command. After the CP TRSOURCE command is completed, any user in the group can enter ETRACE to begin tracing in their own virtual machine, or an authorized user can start ETRACE for the entire group. The data recorded in the system trace file is for the entire group.

You can use the CP TRACERED utility to format and display CP TRSOURCE trace information. The formatted information can either be printed out or placed in a CMS file.

The procedure for setting up and formatting ETRACEs using the CP TRSOURCE command and CP TRACERED utility are found in “External Tracing Facilities” on page 121.

GTRACE

Either the ITRACE or ETRACE command must be entered prior to GTRACE if GTRACE is to work. A GTRACE entry is a special trace entry that can be recorded either internally or externally. It is started by the GTRACE macro, and records up to 256 bytes of application data for an internal trace record and up to 8K for an external trace record. For further explanation of the GTRACE macro, see the *z/VM: Group Control System* book.

Processing Abends

Problems occurring in the system may result in abend (abnormal end) processing. When an abend occurs, an abend completion code is given, an abend work area is filled in, and a dump is taken if DUMP is specified in the ABEND macro. Internal abends always specify DUMP. See “The ABEND DUMP Macro” on page 131 for information on the precedence of SET DUMP.

Abend completion codes give the user some idea of why the error occurred and what part of the system may be responsible for the problem. These codes are explained in the *z/VM: CP Messages and Codes* book.

The abend dump contains information that enables the problem to be tracked further. Using the Dump Viewing Facility REGS command, the contents of the registers at the time the abend occurred can be displayed. The internal trace table and system control blocks can also be displayed. They aid in problem determination and debugging.

The abend work area is used during abend processing to save information about the system at the time of the abend. It contains information such as the registers, the PSW, and the pointer to the next available trace table entry at the time abend occurred. The abend work area address is located at offset X'298' in NUCON.

The Abend Work Area

The abend work area is used during abend processing to save information about the system at the time of the abend.

The abend work area contains the following information:

- The general purpose registers and access registers at the time of error
- The PSW at the time of error
- An abend completion code
- A reason code (if applicable).

It also contains the address of the next available trace table entry at the time the abend occurred.

The trace table entries before this address show the events that preceded the error.

Note: It is possible that an abend can be issued while another abend is being processed. In this case, an abend recursion message is issued.

The recursive abend appears in the trace table. The trace table has recorded the events for both abends.

The abend work area contains information from the original abend, and only the original abend state block (STBLK) (type SVC) remains on the state block chain (see “State Block” on page 141 for information about state blocks).

For abends that result from a program check, the abend work area contains the registers and PSW at the time of the program check.

The field NUCABW in the NUCON (at displacement X'298') points to the abend work area.

The abend work area contains the following important fields:

Displacement	Field Description
X'00' to X'3F'	Registers at the time of failure (0 to 15)
X'40'	PSW at the time of failure
X'48'	Abend code at the time of failure (full word)
X'4C'	Reason code at the time of failure (full word)
X'D8'	Trace pointer at abend time (full word)

X'DC' to X'11B'	Access Registers at the time of failure (0 to 15)

Program Checks

When a program check occurs, an abend results. The abend work area contains the registers and PSW at the time of the program check.

Processing GCS Dumps with the Dump Viewing Facility

The Dump Viewing Facility is a facility that lets you view VM dumps. You should be familiar with the facility and how it works before using the Dump Viewing Facility for GCS dumps.

All dumps taken in GCS are in VMDUMP format and can be viewed using the Dump Viewing Facility.

The Dump Viewing Facility component of z/VM has some DUMPSCAN subcommands you can use to display certain areas of a GCS dump. You must have the GCS nucleus load map in order to use the DUMPSCAN subcommands that are relevant for GCS dumps.

These DUMPSCAN subcommands are:

- DUMPID—displays the dump identifier specified in the SDUMP or SDUMPX commands
- IUCV—displays the entire IUCV path table
- TACTIVE—displays information about active programs on a specified task
- TLOADL—displays the load list for a specified task
- TSAB—displays the task storage anchor block for a specified task
- VMLOADL—displays information about all programs loaded in virtual storage.

You can use other Dump Viewing Facility commands with a GCS dump to aid in debugging. Any Dump Viewing Facility command or subcommand that is valid for VM dumps can help with a GCS dump. The PRTDUMP command and the DUMPSCAN subcommands of CHAIN, DISPLAY, and LOCATE are most helpful when debugging with the Dump Viewing Facility. For more information on these commands, see the *z/VM: Dump Viewing Facility* book.

Information Used by the Dump Viewing Facility

The Dump Viewing Facility uses general purpose control blocks.

For more information on abend work areas, see “The Abend Work Area” on page 136. Program management control blocks are displayed by DUMPSCAN subcommands. Those fields are:

- From the virtual machine load list (displayed by the VMLOADL subcommand)
  - The major NUCCBLK address
  - The module name (major NUCCBLK)
  - The entry point address (major NUCCBLK)
  - The module size (major NUCCBLK)
  - The module load address (major NUCCBLK)
  - The minor NUCCBLK address
  - The entry point name (minor NUCCBLK) and
  - The type of minor NUCCBLK (ALIAS or IDENTIFY).

- From the task load list (displayed by the TLOADL subcommand)
  - The task ID
  - The task block address
  - The load block address
  - The module name and
  - The load count.

For more information on program management, see “Program Management” on page 146.

Task management control blocks are displayed by the DUMPSCAN TACTIVE subcommand. The fields are:

- The task ID (TIDTB)
- The task block address (TIDTB)
- The task completion code (TBK)
- The state block address (TBK and STBK)
- The state block type (STBK)
- The state block module name (STBK)
- The state block module entry point address (STBK) and
- The state block general registers (STBK).

For more information on task management, see “Task Management” on page 140.
Storage management control blocks are displayed by the DUMPSCAN TSAB subcommand. The fields are:

- The pointer to the TSABE (TSAB extension), which contains a pointer for each grain of storage to a list of task storage header blocks that describe the storage owned by a task (for terminology see “Storage Management” on page 153).
- A 256-bit map of subpools owned by a task (TSAB).

In addition, the TSAB subcommand also displays for each task:

- The task ID (TBK)
- The task block address (TBK)
- The task storage anchor block address (TBK).

For more information on task management, see “Task Management” on page 140.

IUCV management control blocks are displayed by the DUMPSCAN IUCV subcommand. The fields are:

- The user ID block (UIDB) address (IUCPT)
- The exit address (IUCPT)
- The user word (IUCPT)
- The task block address (IUCPT) and
- Flags of the path status (IUCPT).
- The Dump identifier, if present

For more information on IUCV, see “The Path ID Table (IUCPT)” on page 152.

NUCON and SIE

In GCS, the NUCON control block and the SIE state descriptor block are located in the first virtual page of GCS. Each GCS virtual machine, when logged on, has its own NUCON and SIE.

There may be times when diagnosing problems on a running system may be preferable to looking at a dump. In these cases the QUERY ADDRESS command can often make chaining through control blocks and data areas easier. Refer to the *z/VM: Group Control System* book for more information on this command.

The data contained in these two blocks is not shared, as the various fields in the NUCON and SIE relate to the operation of a specific user rather than the group.

The NUCON contains many important fields describing the current status of GCS in a GCS virtual machine. Examples of such fields are:

- The various old and new program status words (PSWs)
- The I/O subsystem identification word (SID) (X'B8' in the NUCON)
- The I/O interrupt parameter (X'BC' in the NUCON)
- The virtual machine's user ID (X'204' in the NUCON)
- The task ID of the currently active task (X'212' in the NUCON)
- A pointer to a string of the four anchors of common storage: low common start, low common length, high common start, and high common length.

In addition, other important GCS control blocks are pointed to by NUCON fields. Examples of those control blocks are:

- The task block of the currently active task (pointed to from X'214' in the NUCON)
- The common trace block (pointed to from X'21C' in the NUCON)
- The SIE (pointed to from X'5C4' in the NUCON)
- Various work areas (for example, the abend work area, pointed to from X'298' in the NUCON).

The SIE is an extension of the NUCON and contains further pointers to other control blocks. Some pointers, useful when performing diagnostics, that you can find in the SIE are:

- The address of the task ID table (X'10' in the SIE)
- The address of the asynchronous exit queue (X'18' in the SIE)
- The address of the virtual machine control block (VMCB) (X'2C' in the SIE)
- The address of the storage management anchor block (SMAB) (X'40' in the SIE).

Virtual Machine Control Block

When a virtual machine IPLs GCS, a VMCB is maintained for that machine. There are as many VMCBs as the maximum number of virtual machines that can join the GCS group (the maximum number is specified at GCS generation time).

A VMCB is 24 bytes long and, among other information, contains:

- The virtual machine user ID (the first 8 bytes of the VMCB)
- The machine ID (the 2 bytes at displacement X'0A' of the VMCB).

For other information on VMCBs, see "VMCB—Virtual Machine Control Block" on page 212.

How to Determine the User ID That Created a Trace Entry

Each entry in the GCS internal trace table has a reference to the **machine ID** of the virtual machine that created the entry. The machine ID is a binary number assigned to the virtual machine when GCS is IPLed in the virtual machine.

To determine the user ID that created a trace entry, you have to translate the machine ID to its corresponding user ID. In other words, you have to access the VMCBs of the GCS virtual machine, because the VMCB is the place where user ID and machine ID are correlated.

To find the VMCBs of the virtual machines in a GCS group, use the following procedure:

1. Locate the SI Extension (SIE) address in the NUCON at X'5C4'
2. Find the address of the VMCB array at SIE + X'28'.

How to Locate the GCS Common Lock

The SIELKCOM field in the SIE (at displacement X'20') points to the common lock. The common lock is a word-long field in common storage that contains the machine ID (2 bytes) and the task ID (2 bytes) that are currently holding the common lock. If the common lock is free, it contains binary zeros.

The GCS QUERY LOCK command can be used to display the status of the common lock. A query on the lock is sufficient to determine if the lock has changed since the last query.

When you are recreating a problem and you want to know when the common lock is being acquired, use the CP TRACE command. This can be done by entering a CP TRACE on a store into the common lock word, and when CP TRACE stops the virtual machine you can display the machine and task ID values.

If at that time you take a dump of the virtual machine that has acquired the lock, you will be able to use DUMPSCAN subcommands to interrogate the task in question and determine what module is issuing the request for the lock.

An alternative could be to use the CP TRACE command to display stores in the SVC OLD PSW (at displacement X'20' in the NUCON). This would be an SVC 203 (X'CB') for the LOCKWD macro.

The mapping of the NUCON in GCS is different from that in CMS. The SIE has also been added as an extension of the NUCON. Both contain important information for the debugging of GCS. For more information on the NUCON and SIE Extension mappings and field descriptions, see “NUCON—GCS Nucleus Constant Area” on page 199 and “SIE—NUCON Extension” on page 203.

Task Management

Task Block

The task block (TBK) gives you a good idea of the state of a task. The task block for a task is pointed to from the task ID table and contains information such as:

- A pointer to a list of state blocks describing the programs that have been running under the task

- A pointer to a list of load blocks describing the programs that the task has loaded in storage through a LOAD SVC (SVC 8 or SVC 122) or through the GCS LOADCMD command
- The value of the registers and PSW when the task is dispatched, if the task is dispatchable
- The address of the task block of the owning task
- The task ID and task priority.

For information on the task block mapping and field descriptions, see “TBK—Task Block” on page 205.

State Block

GCS uses state blocks (STBLKs) to keep track of a particular task’s processing activity.

There is a state block for each active program in the task. The primary purpose of the state block is to save and restore PSWs and other processing status in particular steps in a task.

The chain of state blocks for a task can be seen as an *active stack*:

- When the task is created, a state block for that task is also created. This state block is always called INIT.
- When certain events occur in the task, GCS adds new state blocks to the top of the stack. GCS sets a flag byte (at displacement X'24') in the state blocks to indicate what type of event has occurred:
  - If the task has issued a LINK, SYNCH, XCTL, or ATTACH macro, the flags contain X'80', and the state block is referred to as a LINK block.

    Note: If the task has issued a SYNCH macro with RESTORE=YES, the flags contain X'90'. The RESTORE=YES operand tells GCS that the general registers 2 through 13 are to be restored when control is passed back to the calling program.
  - If the task has issued an SVC instruction, the flags contain X'40', and the state block is referred to as an SVC block.
  - If an asynchronous exit has been scheduled for the task, the flags contain X'20', and the state block is referred to as an AEB block.

    In this case, other flags (at displacement X'25') in the AEB block, indicate whether the asynchronous exit was scheduled as a result of a SCHEDEX macro, an I/O interrupt from a general I/O device, or a timer interrupt.
- When a program represented by a state block ends, the corresponding state block is removed from the top of the stack.

The preceding discussion leads to the conclusion that the analysis of the existing state block chain (stack) for a task gives an important idea of the events (LINK, SVC, or AEB) that are still being handled, and the order they have occurred.

The state block chain is pointed to from the task block with the most recently added state block at the beginning of the chain.

The PSW and the general registers in a state block refer to the previous program running under the state block. The PSW for a running program is in the task block.

For more information on the state block mapping and field descriptions, see “STBLK—State Block” on page 207.

WAIT COUNT Field in a State Block

An important field in a state block is WAIT COUNT. Use this field (STBWAIT at displacement X'26' in the state block) to determine if a task is waiting.

If the contents of the field are:

Value	Meaning
0	The task is not in a wait state.
1	The task is in a wait state.

Note that the STBWAIT field is maintained by GCS only if the task has used the WAIT SVC (SVC 1) to enter a wait state.

By looking into the instructions that precede the SVC instruction, you probably will find a LOAD (L) or a LOAD ADDRESS (LA) instruction that loads in Register 1 the address of the ECB (Event Control Block) (or ECBLIST) associated with the wait. Use this to determine what the task is waiting for.

Note: If the task has entered a wait state by other means (for example, by a LOAD PSW instruction, if the task was running in supervisor state) this is not reflected in the STBWAIT field.

LINK Block

A LINK block is a type of state block that represents a module to which control was passed when the task issued a LINK, SYNCH, XCTL, or ATTACH macro.

When that module returns control to the program that issued the macro, the LINK block is removed from the state block chain of the task.

The caller’s registers are not moved into a LINK block unless it is for a SYNCH macro with RESTORE=YES.

The second word of the PSW in the LINK block (field STBPSW) points to the address following the SVC instruction. Use this address to determine the module that has issued the ATTACH, LINK, SYNCH, or XCTL macro.

SVC Block

An SVC block is a type of state block that represents a module to which control was passed when the task issued an SVC instruction.

The second word of the PSW in the SVC block (field STBPSW) points to the address following the SVC instruction. Use this address to determine the module that has issued the SVC instruction.

Asynchronous Exit Block (AEB)

The AEB is a type of state block that represents an asynchronous exit that has been scheduled to be run under a task.

Certain flags in an AEB indicate whether the asynchronous exit has been scheduled by general I/O, SCHEDEX, or TIMER functions.

When an asynchronous exit is to be scheduled to run under a task, GCS gets an AEB from storage, fills in the appropriate fields—such as register values, task block address the AEB is to run under, and the entry point of the exit routine—and queues that AEB on the SIEAEQ. It is then dispatched from the SIEAEQ to the appropriate task state block chain.

Asynchronous exits resulting from SCHEDEX functions have their AEB blocks in two additional chains:

SIEAEQ
Is a field in the GCS SIE control block that contains a pointer to a queue of AEBs (located in private storage), to run in a virtual machine. This queue is used as follows:

1. When a task, A, in a virtual machine wants to schedule an exit to run in another task, B, task A issues the GCS SCHEDEX macro, specifying the task ID of task B and the exit routine address.
2. GCS SCHEDEX processing, running for the “SCHEDEXing” task, gets an AEB, fills in the appropriate fields, and queues the AEB in the SIEAEQ.
3. When the GCS dispatcher gets its turn to run, before dispatching any tasks, it checks if there are any AEBs queued in the SIEAEQ.

   If so, it takes the AEB off the SIEAEQ and queues it at the beginning of the task B state block chain.
4. When task B eventually gets dispatched, the exit routine runs as the currently active state block.

VMCSCHDX
Is a field in the virtual machine control block (VMCB) that contains a pointer to a queue of AEBs (located in common storage) used in cross-machine exit functions. The pointer to VMCB is in the NUCON (SIE at displacement X'28'). For more information on VMCB, see “VMCB—Virtual Machine Control Block” on page 212. An example of how this queue is used is:

1. When a task, A, in the virtual machine A wants to schedule an exit routine to run in a task B in the virtual machine B, task A issues the GCS SCHEDEX macro, specifying the machine ID of virtual machine B, the task ID of task B, and the exit routine address.
2. GCS SCHEDEX processing, running for the SCHEDEX task, gets an AEB, fills in the appropriate fields and, using Compare/Swap logic, queues the AEB on the VMCSCHDX queue associated with the target virtual machine (B).
3. After that, GCS running in virtual machine A issues an IUCV message to virtual machine B that informs it about the exit routine to be scheduled.
4. Virtual machine B is interrupted by the IUCV message (external interrupt).
5. The IUCV interrupt handler in GCS calls the GCS scheduling routines GCTSDT and GCTSDX.

   These routines find the VMCB of the virtual machine B, dequeue any AEBs queued on the VMCSCHDX queue for this virtual machine, and queue them in the SIEAEQ queue.
6. Finally, when the dispatcher gets control in virtual machine B, before dispatching any tasks, it checks if there are any AEBs queued in the SIEAEQ.

If so, it takes the AEB off the SIEAEQ and queues it at the beginning of task B State block chain.

7. When task B eventually gets dispatched, the exit routine runs as the currently active state block.

The Dispatch Queue

Because GCS is a multitasking environment, tasks are performed concurrently. The dispatcher is called each time a new task can be run. System services, such as interrupts and service calls (SVCs), pass control to the GCS dispatcher.

Within a virtual machine there are multiple tasks to perform. Each task has a priority associated with it. The task with the highest priority is given control to run first.

To keep track of tasks and their priorities, a dispatch queue is set up which chains the tasks (through task blocks) by priority. The task with the highest priority is placed at the beginning of the chain. Each priority level contains tasks of equal priority. Each level is capable of containing more then one task, but each task on that level is of the same priority.

If a task has been running an extended amount of time, the dispatcher switches to another task of equal priority that is waiting in the dispatch queue. This only happens if there is a task of equal or higher priority waiting to be processed.

When the dispatcher is ready to dispatch a task, it first looks at the tasks with the highest priority level. These tasks are at the beginning of the dispatch queue. If the first task on that level is ready to run, it is given control. If not, the next task (if any) on the same priority level is checked.

This is continued until a task is found ready to run. If no tasks on that priority level are ready to run, the next priority level is checked until a ready to run task is found.

To find and follow the dispatch queue:

1. Locate the SI extension (SIE) address in the NUCON at X'5C4'.
2. Find the address of the first task block (TBK) on the dispatch queue at SIE + X'14'.
3. TBK + X'00' is the address of the task block on the dispatch queue of higher priority than this task block.
4. TBK + X'04' is the address of the task block on the dispatch queue of lower priority than this task block.
5. TBK + X'08' is the address of the next task block of the same priority.
6. TBK + X'C' is the address of the previous task block of the same priority.

   All of the task blocks on this chain are of the same priority and are dispatched in turn.

Using the steps listed, the whole dispatch queue can be traversed and each task waiting to be run can be found.

For more information on the Dump Viewing Facility and task management control blocks, see “Processing GCS Dumps with the Dump Viewing Facility” on page 137.

How to Find the Task ID Table

The task ID table lists all the tasks in the virtual machine. All valid task blocks (TBK) are anchored in the task ID table (TIDTB). This table can be used to find all tasks or a specific task by its ID. The make-up of the task ID table is shown in Figure 10.

To find the task ID table (TIDTB):

- Locate the SI extension (SIE) address in the NUCON at X'5C4'.
- Find the TIDTB address at SIE + X'10'.
- The first 8 bytes in the table are table control data and do not point to a task block. Instead it contains a table label and a pointer to the next task ID table.
- The table entries start at the TIDTB address + X'08'.
- Each TIDTB has 255 entries.

Each TIDTB entry describes a task:

- Each entry is 8 bytes long
- The first halfword (the first 2 bytes) in an entry contains the task ID
- The following halfword (the second 2 bytes) is unused
- The next fullword (the last 4 bytes) contains the address of the task block for that task.

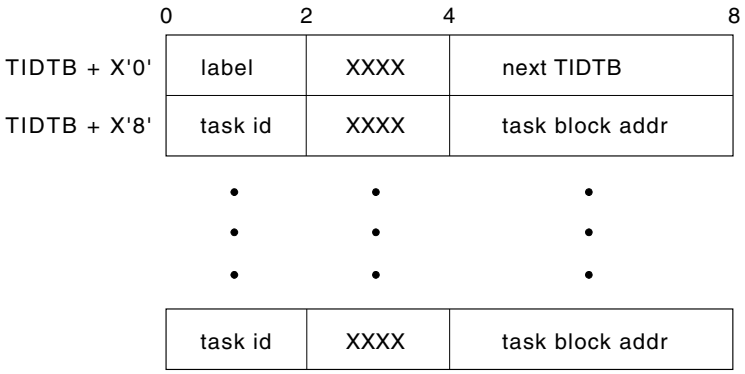


Figure 10. The Task ID Table (TIDTB)

How to Find Which Task Is Running

In NUCON there is a field that contains the task ID of the task currently running. Use this task ID and find its entry in the task ID table. In NUCON there also is a field that points directly to the task block (TBK) of the task currently running. This address and the address of the task block in the TIDTB for the current task ID should be the same.

- Locate the active TBK address in the NUCON at X'214'.
- Locate the address of the state block of the last active module at TBK + X'10'.

See “TBK—Task Block” on page 205 and “STBLK—State Block” on page 207 for important fields.

If you are using the Dump Viewing Facility, the following procedure using DUMPSCAN subcommands yield similar information in formatted form:

- Enter DISPLAY X'212' to get the current task ID.
- Enter TACTIVE using the task ID just found.

- The display that results includes the completion code, the program name, and the register contents associated with the state block.

If an abend occurs with a dump while GCS is processing an I/O interrupt or an external interrupt, the pointer to the active task will point to a special task block located in low storage. It will not be on any task block chain. The X'02' flag at displacement X'CE' will be on to signify that is the special interrupt task block. See displacement X'CE' and its flag bytes and descriptions page 205.

Tracing Task and Program Management

ITRACE and ETRACE facilities record supervisor events and the GTRACE macro records user events, as these events occur in GCS. Included in these event recordings are the dispatcher and program interrupt trace table entries. These entries can be of use when debugging potential task and program management problems.

- The dispatcher trace entry (X'01' type) is made whenever a task is dispatched. If an active task is being redispatched, no trace table entry is created. The entry includes the task ID, task block address, and PSW.
- The program interrupt entry (X'04' type) is made each time a program interrupt occurs. It includes such information as the task ID and program old PSW.
- Each GTRACE entry in the trace table includes the task ID of the task that issued the GTRACE.

Program Management

When you are analyzing a dump of a GCS virtual machine, there are some important control blocks that give you information about the programs loaded in storage.

A program can use GCS program management macros to dynamically load and run program modules by name. GCS macros that may cause GCS to load a module in storage are:

LOAD Loads a module into storage. Control is not passed to the loaded module.

LINK Loads a module and calls it. When the LINKed module returns, control is also returned to the module that issued the LINK.

XCTL Loads a module and transfers control to it. When the XCTLed module returns, control is not returned to the module that issued the XCTL. Instead, control is passed to the module that called the issuer of the XCTL macro (if there is one) or to GCS.

The above GCS macros refer to a module by its entry point name (or ALIAS entry point name as defined in the LOADLIB libraries).

When looking for the entry point, GCS searches the following items in sequence:

1. The **virtual machine private storage**, because the module associated with the entry point name may already be loaded.
2. Any **saved segment directories** that may have been created with the GCS CONTENTS macro, which sets up a directory for the entry points in that segment.

For example, the VTAM saved segment has a directory built with the CONTENTS macro. Therefore, you are able to LINK, LOAD, and XCTL to the VTAM entry points.

The VSAM saved segment (used by NetView®) does not have a built-in directory. Therefore, you are not able to LINK, LOAD, and XCTL to the VSAM entry points.

3. The **directories of any load libraries** that may have been defined for the virtual machine through the GCS GLOBAL LOADLIB command.

If the module cannot be found in storage and it exists in a load library, GCS loads the module into storage. GCS keeps track of modules loaded in storage through two lists:

- The **virtual machine load list**, which describes all the modules that have been loaded into storage.
- The **task load list**, which associates loaded modules with the task that caused the module to be loaded. Note that only modules for which the task has issued a LOAD SVC are referred to in the task load list.

  Note: In addition to this list, GCS also creates a state block for a task each time the task issues the ATTACH, LINK, SYNCH, or XCTL macro. State blocks are discussed in "State Block" on page 141.

In addition, other GCS macros are used with the program management functions:

IDENTIFY
Allows dynamic creation of a new entry point for a loaded module.

SYNCH
Calls a loaded module.

DELETE
Removes a module from storage.

BLDL Requests GCS to locate a module in a GLOBALed LOADLIB and to retrieve the module size and characteristics.

Task Load List

The task load list is made up of load blocks representing programs that a task has requested through the LOAD macro. There may be a load list for each task. The load list consists of load blocks chained together and pointed to by the task block (TBK + X'14').

The load block (LDBLK) contains the following information:

Displacement	Field Description
X'00'	The program name
X'08'	The address of next load block on chain
X'0C'	The address of previous load block on chain
X'10'	The address of NUCCBLK for this load block
X'14'	The load count (2 bytes)
X'16'	Flag
	Byte **Field Description** **1xxx** Load issued by LOADCMD
X'17'	RMODE and AMODE

You may enter a LOAD for a program more than once. The load count keeps track of the number of LOADs issued for a program by a particular task. The count ensures that the storage used to load the program is not freed while being used by the program. The LOADCMD flag is used ensuring that the program storage is not freed at command termination. For more information on the LOADCMD command, see “LOADCMD Command” on page 171.

Virtual Machine Load List

When GCS loads a program into storage, it builds a major NUCCBLK that contains information about the program that was loaded. When a task issues a LOAD, LINK or XCTL macro for a module that exists in the shared segment directory, GCS builds a major NUCCBLK. If the loaded entry point is an ALIAS entry point, or if an IDENTIFY macro is issued for a loaded program, GCS builds a minor NUCCBLK. The minor NUCCBLKs are chained together and pointed to by the corresponding major NUCCBLK. When the major NUCCBLK is deleted, the minor NUCCBLKs associated with it are also deleted.

The list of major NUCCBLKs is pointed to from the field NUCCBLKS in the NUCON (at displacement X'5E0'). The NUCCBLKs contain the following information:

Displacement	Field Description
X'00'	The program/alias/identify name
X'08'	The next NUCCBLK
X'0C'	The previous NUCCBLK address for the major NUCCBLK or major NUCCBLK address for the minor NUCCBLK
X'10'	The entry point address
X'14'	Flags
X'16'	The use count for the major NUCCBLK
X'18'	Key
X'19'	AMODE and RMODE from the LOADLIB **X'10'** RMODE ANY **X'03'** AMODE ANY **X'02'** AMODE 31 **X'01'** AMODE 24
(Major NUCCBLK only)	
X'20'	The program start address or zero
X'24'	The program size or zero
X'28'	The alias / minor NUCCBLK address

The above maps both a major and a minor NUCCBLK. The major NUCCBLK is larger with the additional fields at the end of the block. The program start address and size will be zero if the program resides in common storage or a shared segment. The KEY is filled in only for a major NUCCBLK and is the first bits in the field.

The FLAGS field is 2 bytes long and is used as follows:

Byte	Field Description
First Byte:	
1xxx xxxx	A major NUCCBLK
x1xx xxxx	An alias minor NUCCBLK
xx1x xxxx	An identify minor NUCCBLK

Second Byte:	(Only used in the major NUCCBLK)
1xxx xxxx	Reentrant
x1xx xxxx	Reusable
xx1x xxxx	A reusable module and currently in use
xxx1 xxxx	The module is executable
xxxx 1xxx	In common storage or shared segment
xxxx x1xx	The module is non-reusable and has been used

How to Find Where a Program Is Loaded

Depending on what you know about a program, you can use one of the following methods to find where the program is loaded and other information about the program.

1. If the program you are looking for is running in the current task:
   - Using the procedure given in “How to Find Which Task Is Running” on page 145, find the task block (TBK) for the task ID for the program.
   - After the task block is located, locate the active state stack pointer at TBK + X'10'. This points to the first state block in a chain.
   - Locate the program name in the state block (STB) at X'00'. The program name may be the name of an ALIAS or IDENTIFY as well as the main program itself.
   - If this is not the name of the program you are looking for, follow the state block chain to the next state block. Locate the chain pointer at X'10' in STB.
   - If the program name is ‘INIT ’ or the chain pointer is zero, you have reached the end of the chain. The program being searched for may not be running under this task, or was not called by the program management SVC macros.
   - When the state block for the program is found, locate the address of the NUCCBLK at X'1C' in STB.
   - The NUCCBLK contains information about the program, such as its name, entry point address, where it is loaded, and more.
   - If you only wish to know the entry point address for the program, it can be found in the state block at STB + X'20'.
2. If you know that the program has been loaded using the LOAD macro, and that it has been debugged using the Dump Viewing Facility, you can use the following method to find where the program is loaded.
   - Enter the TLOADL subcommand of DUMPSCAN to display the NUCCBLKs.
   - The resulting display includes the load blocks for the tasks specified when issuing the TLOADL subcommand. Each load block contains the program name and the address of the NUCCBLK. The NUCCBLK contains the address of the loaded program. For more information on NUCCBLK and load blocks see “Task Load List” on page 147 and “Virtual Machine Load List” on page 148.
3. If the following are true:
   - You have the program name
   - The program has been debugged using the Dump Viewing Facility
   - The program has not been loaded by using the LOAD SVC.

   You can use the following method to find where the program is loaded:
   - Enter the VMLOADL subcommand of DUMPSCAN to display the NUCCBLKs.
   - The resulting display includes the major NUCCBLKs and minor NUCCBLKs. The major control blocks represent the module itself, and the minors map

IDENTIFY or ALIAS entry points. The module name and address are found in the major NUCCBLK, and the ALIAS or IDENTIFY entry points are found in the minor NUCCBLK.

- The NUCCBLK addresses are also given in case you wish to display the NUCCBLK in storage for more information about the program.

4. If you have the program name and the program is not running on the active task or you are not debugging in the Dump Viewing Facility, you can follow the chain of NUCCBLKs in the following method:
   - Display the NUCCBLK block address at X'5E0' in the NUCON.
   - Locate the program name at X'00' in the NUCCBLK. If this is not the program name, follow the major NUCCBLK chain to the next NUCCBLK. Locate the chain pointer at X'08' in the NUCCBLK.
   - If the program name may be an ALIAS or IDENTIFY, search through the minor NUCCBLKs before going to the next major NUCCBLK. The pointer to the first minor NUCCBLK is located at major NUCCBLK + X'28'. In the minor NUCCBLK, the chain pointer for minor NUCCBLKs is located at minor NUCCBLK + X'08'. A zero in this field indicates the end of the chain.
   - After the NUCCBLK for the program is found, you can use the information in the NUCCBLK to find out more about the program—the entry point address, where it is loaded, or its size, for example.

GCS Load Error

If your job abends with an abend code of 106 and a reason code of 030B in register 15 when you are loading a module, the GCS abend was caused by a disk I/O error. The reason for the disk I/O error can be found in the ERRCODE field of the DIODA.

To locate the ERRCODE field:

- Locate the address of the DIODA (NUCDIODA) at X'67C' in the NUCON
- ERRCODE is at displacement X'FF' into the DIODA.

IUCV

Note: In the IUCV section, when the word user appears, it refers to any supervisor or problem program.

GCS supports communication within a virtual machine or between any two virtual machines by using IUCV. Routines running within a task communicate through IUCV with one of the following:

- Other routines in the same machine (same task or different task)
- Routines in other virtual machines
- CP.

When communication is set up through IUCV, the user is assigned a linkage for communication called a **path**. A path is established when the source communicator calls the IUCV CONNECT function using the IUCVCOM macro, and the target communicator calls the IUCV ACCEPT function, again using the IUCVCOM macro. Both the source and target communicators must be defined in the GCS IUCV environment for a path to be established between them. That is, each must issue an IUCVINI SET macro function first.

A single communicator can have multiple paths defined at a time. When an IUCVINI SET macro is issued to admit a user into the IUCV environment, an authorized user may make himself privileged, using the PRIV=YES parameter if the user is running in supervisor state. This lets the task communicate on a path using IUCV directly, rather than through the GCS IUCV support.

For more information on IUCV, see the *z/VM: CMS Application Development Guide for Assembler* book. GCS IUCV support is further discussed in the *z/VM: Group Control System* book.

Debugging Applications

When IUCV problems are first suspected, you should ensure that the application or program running is using IUCV correctly and that the parameter lists are set up correctly. TRACE stops should be set after IUCV macros are issued within a program or application. After the IUCV function has completed, check the return code in register 15 and any other information that is returned in the CP IUCV parameter list. If the return code in register 15 is over 1000 (decimal), the error occurred while the IUCV function was being processed by CP. The IPRCODE field in the CP IUCV parameter list indicates the cause of the error.

Tracing IUCV

IUCV can be traced through the trace facility. Both CP and GCS keep track of IUCV with trace table entries. CP trace makes an entry into the CP trace table for each IUCV function that it processes. ITRACE and ETRACE make IUCV trace table entries each time an IUCV SVC or external interrupt occurs for GCS. For more information on GCS Trace facilities see “Using the GCS Trace Facilities” on page 132.

The IUCV Anchor Block (IUCBK)

The IUCV anchor block (IUCBK) contains general information about the GCS IUCV environment. It is pointed to from the SIE at SIE + X'B8'.

The IUCV anchor block contains the following among other information:

Disp	Label	Field Description
X'00'	IUCCBFAD	Address of control external interrupt buffer (EIB)
X'04'	IUCEIBAD	Address of application external interrupt buffer
X'08'	IUCVIDAN	Address of user ID block (IUCID) chain
X'0C'	IUCPRMAD	Address of internal copy of IUCV parameter list
X'10'	IUCVPTAD	Address of path ID table
X'14'	IUCVSAVE	Address of user savearea
X'24'	IUCVCONN	Maximum number of connections allowed (from MAXCONN in VM directory entry)

The control external interrupt buffer (EIB) contains information about the last interrupt on a control path. The application EIB contains information about the last interrupt on a non-control path. For more information about control paths see the *z/VM: CP Programming Services* book.

The user ID block (IUCD) chain and the path ID table are explained later in this chapter in more detail.

The IUCPRMAD points to a copy of the last CP IUCV parameter list that was issued by the GCS IUCV support, either implicitly (IUCVINI) or explicitly

(IUCVCOM). The internal parameter list holds a copy of the last CP IUCV parameter list that was issued by the GCS IUCV support on behalf of one of its users. It is also used for IUCV functions that GCS IUCV support must start, for example, to sever an incoming path to a user that has not issued an IUCVINI SET function.

The User ID Blocks (IUCID)

User ID blocks contain information about active users in the IUCV environment. There is an IUCID for each user, containing the user name, user word, and associated task block address. The IUCIDs are chained together, with the most recently added user at the beginning of the chain. The first IUCID is pointed to by IUCVIDAN in the IUCV anchor block (IUCBK).

The user ID block is built when a user is admitted into the IUCV environment using the IUCVINI SET macro. The name specified in the macro is the name by which the user is known in the IUCV environment. When paths are established using IUCVCOM CONNECT and IUCVCOM ACCEPT functions, the user names specified on the two macro invocations identify the two parties wishing to do IUCV communications. The IUCVINI CLR macro ends the IUCV environment for the specified user. When the user is terminated from IUCV, the associated user ID block is deleted from the user ID chain, and all paths for the user are severed.

The IUDB contains the following information:

Displacement	Field Description
X'00'	The next user ID block address
X'04'	The general exit address
X'08'	The user name
X'10'	The user word
X'14'	The task block address
X'18'	Flags

Byte	Field Description
1xxx	The problem state indicator
x1xx	The privilege state indicator
xx1x	The exit will be run in AMODE 31

The Path ID Table (IUCPT)

The path ID table contains an entry for every possible IUCV path based on the maximum number of paths available for this virtual machine. A path entry is filled in when the path is established using IUCVCOM CONNECT, and also on the resulting pending connect interrupt. Therefore, a single communication's path is represented by two path entries. A path can be in different states as indicated by the flags in the path entry. Before any GCS IUCV function is processed, the state of the path is checked to see if the function is allowed.

For more information on the Dump Viewing Facility and IUCV management control blocks, see "Processing GCS Dumps with the Dump Viewing Facility" on page 137.

Each path ID table entry is 20 (X'14') bytes long.

The path ID table contains the following information:

Displacement	Field Description

X'00'	The address of user ID block
X'04'	The exit address
X'08'	The user word
X'0C'	The task block address
X'10'	Flags

Byte	Field Description
1xxx xxxx	The path is active.
x1xx xxxx	The connect is issued.
xx1x xxxx	The connect is pending.
xxx1 xxxx	The path is quiesced.
xxxx 1xxx	The path is severed.
xxxx x1xx	The exit will be run in AMODE 31
xxxx xx1x	Problem state indicator
xxxx xxx1	Privilege state indicator

The task block address represents the task that was running when the path was created. The user ID block address points to the user ID block for the owner of the path. The exit address is for the owner's path-specific exit.

How to Find Information about a Path

You can find information about a path, such as who owns it and its present status, in a path ID table entry for the path. The path ID provides an index into the path table to get to the entry that describes the particular path.

- If you have a VMDUMP formatted dump, you can use the Dump Viewing Facility.
  - Enter the Dump Viewing Facility DUMPSCAN IUCV subcommand
  - The resulting display shows the important information found in each of the path entries in the path ID table.
- If you are manually displaying addresses and following chains, this procedure yields the path table entry for a specific path ID:
  - Locate the SI extension (SIE) address in the NUCON at X'5C4'.
  - Locate the IUCV anchor block (IUCBK) address at SIE + X'B8'.
  - Locate the path ID table (PIDT) address at IUCBK + X'10'.
  - The specified path ID is in hexadecimal.
  - Calculate the offset as follows:

    Offset = pathid x X'14'.

    Each path table entry is X'14' or 20 bytes long.

    For example, if pathid = X'B', the path entry is at displacement X'B' x X'14' = X'DC' into the table.
  - The path entry is located at PIDT + offset.
  - See the path ID table entry map for the layout of the path entry.

Storage Management

The storage management component of GCS controls the allocation of storage for a GCS virtual machine. GCS manages storage with three different perspectives:

- Storage location (private or common storage, above or below the 16 megabyte line)
- Storage protection (storage key and fetch or store protection bits)
- Storage ownership (persistent or task related storage).

Information about common storage for the whole virtual machine is in the storage management anchor block (SMAB). To locate the SMAB, first locate the SIE address at location X'5C4' in the NUCON and then locate the SMAB at displacement X'40' in the SIE.

The fields describing common storage are:

1. The address of the start of low common storage is in SMASCOML (SMAB + X'60').

   The length of low common storage is in SMALCOML (SMAB + X'64').
2. The address of the start of high common storage is in SMASCOMH (SMAB + X'68').

   The length of high common storage is in SMALCOMH (SMAB + X'6C').

For more information on storage management mapping and field descriptions, see "SMAB—Storage Management" on page 209.

Storage Anchor Blocks

There are five types of storage anchor blocks:

- Private storage anchor blocks:
  - Low private anchor block (LPAB)
  - High private anchor block (HPAB),

  depending on the position of the private storage—above or below the 16 megabyte line.
- Common storage anchor blocks:
  - Low common anchor block (LCAB) and
  - High common anchor block (HCAB),

  depending on the position of the common storage—above or below the 16 megabyte line.
- Task storage anchor blocks (TSAB).

The first four storage anchor blocks (LPAB, HPAB, LCAB and HCAB) are identical. They contain pointers to the start of arrays of major and minor storage anchor control blocks (SACBs) describing the free storage pages.

The TSAB contains a pointer to the TSAB extension which is a string of pointers to the start of a double-linked list of task storage header blocks (TSHBs), one pointer for each division (or grain) of the storage. The TSHBs describe the storage belonging to a task.

To find any of the four free storage anchor blocks:

1. Locate the SIE address at displacement X'5C4' in the NUCON.
2. Locate the pointer to the storage management anchor block (SMAB). This pointer is at displacement X'40' in the SIE.
3. The LCAB is pointed to by the SMALCAB field (at SMAB + X'00').
4. The HCAB is pointed to by the SMAHCAB field (at SMAB + X'04').
5. The LPAB is pointed to by the SMALPAB field (at SMAB + X'08').
6. The HPAB is pointed to by the SMAHPAB field (at SMAB + X'0C').

The TSAB is pointed to by the field TBKSTOR at displacement X'A8' in the task block (TBK).

For more information on the storage anchor block mapping and field descriptions, see “ANCH—Storage Anchor Block” on page 210.

Description of the Storage Anchor Control Blocks (SACBs)

There are two types of storage anchor control blocks (SACBs): major and minor.

A major SACB is 14 bytes long, and a minor SACB 10. They are in contiguous storage, are built at initialization time, and are permanent.

There is a major SACB to describe each page of free storage. Contiguous to each major SACB is a chain of minor SACBs. Each of these describes a noncontiguous free area in the page.

Important Fields in Major SACBs

The major SACBs contain the following fields:

Displacement	Field Description
X'00'	MAJNXTPT points to the major SACB for the next page of the same key.
X'04'	MAJBKPTR points to the major SACB for the previous page of the same key.
X'08'	MAJMAXLN is a 2-byte field that names the largest free area on the page that does not begin on a page boundary.
X'0A'	MAJLNCON is a 2-byte field that gives the length of the free area at top of the page.
X'0C'	MAJKEY is an 8-bit field that contains the key and fetch bit for the page.
X'0D'	Flags

Byte	Field Description
1111 xxxx	Not used
xxxx 1xxx	MAJTOLIN SACB to go to no key queue
xxxx x1xx	MAJLIMBO SCAB to go to no key queue
xxxx xx1x	MAJENDL Major SACB is at the low end of array of majors
xxxx xxx1	MAJENDH Major SACB is at the high end of array of majors

Important Fields in Minor SACBs

Minor SACBs are control blocks used for the following purposes and contain specific fields:

1. Combined with a major SACB, they describe free storage on a page boundary. Each of these minor SACBs are headers for a chain of minor SACBs that describe all free storage on a given page.

Displacement	Field Description
X'00'	MNORNXT points to the next minor SACB used to describe the next noncontiguous free area on the same page.

X'04'	MNORPTRF points to the free area on the page boundary.
X'08'	MNORLN is the length of free area on the page boundary; this field has a length of 2 bytes.

2. They describe free storage not on a page boundary. These minor SACBs are found on pages of storage that are chained together and are pointed to by ANCHPGMN in the anchor block.

Displacement	Field Description
X'00'	MNORNXT points to the next minor SACB used to describe the next noncontiguous free area on the same page.
X'04'	MNORPTRF points to free storage not on a page boundary.
X'08'	MNORLN is the length of the free storage, this field has a length of two bytes.

For more information on the Dump Viewing Facility and storage management control blocks, see “Processing GCS Dumps with the Dump Viewing Facility” on page 137.

Checking for Storage Fragmentation

Check the fields ANCHPGL and ANCHPGH, which point to the major SACBs that represent the lowest and highest completely free pages of storage. If these pointers are both zero, then storage is fragmented down to the page level. If they are not zero but the request is for greater than a page, scan the major SACB between these major SACBs to see if there is sufficient storage.

Scanning the Major and Minor SACBs

1. Find the appropriate anchor block for private or common storage.
2. Starting with ANCHMAJL, scan the major/minor combinations:
   a. Major SACBs exist for each page of private/common free storage.
   b. Minor SACBs have the address of the page represented (MINPTRF at X'04').
   c. Match the page represented with the address of the storage in question.
      1) These minor SACBs are contiguous with the major SACBs they describe.
      2) Scroll until the corresponding page is found.

Checking Free Storage on Any Given Page

1. Find the appropriate anchor block for the private or common storage.
2. Starting with ANCHMAJL, scan the major/minor combinations for the major SACB for the appropriate page. For more information, see “Scanning the Major and Minor SACBs.”
3. The first minor SACB is the header for a chain of minor SACBs that describe all free storage for the page. This minor SACB describes the free storage on the lower page boundary. If MNORLN is 4 KB, the page is fully free and is available for use in any key.
4. If MNORLN is not 4 KB, look at MAJMAXLN. This field tells you the largest free piece of storage available on the page not on a page boundary.

Note: Because this page is not completely free, it cannot be used for a request of another key.

5. To calculate free storage for two or more contiguous pages, check MAJLNCON for free storage at the top of the page and MNORLN for free storage at the bottom of the page.
6. To find the description of all free storage on a given page, follow the chain of minor SACBs.

Finding the Key for a Given Page

- To find the actual key for a given page of storage, use the CP command DISPLAY K.
- To see what key GCS has for the same page:
  1. Scan the chain of major SACBs for the one that describes the page you are interested in. For more information, see “Scanning the Major and Minor SACBs” on page 156.
  2. To find the key and fetch bit in MAJKEY:
     a. The GCS storage management key and fetch protect bit are right-justified.
     b. In GCS, 1C corresponds to E0 through E7 in CP, meaning key 14 nonfetch-protected storage.

MAJKEY	000kkkkF	CP KEY	KKKKFXXX

- To check pages of free storage in any given key and fetch protection:
  1. Find the appropriate anchor block for private or common storage.
  2. ANCHKEYP (at X'04' in LPAB or HPAB) is the start of an array of 32 records that are the anchors for chains of major SACBs for each key and protection status.
  3. To find the appropriate pointer for the key and fetch protection you want to follow down the chain:
     a. The first pointer is for key zero nonfetch-protected, the second for key zero fetch-protected, and so on.
     b. This pointer will point to the first major SACB that describes free storage for the key and fetch protection.
     c. Use MAJNXTPT, the forward pointer, and MAJBKPRT, the backward pointer, to follow up and down the chain.

Control Blocks Describing the Storage Owned by a Task

Task-owned storage can only be in **private** storage. Though a task can get **common** storage with the GETMAIN macro, that storage is not automatically freed when the task ends and must be freed with the FREEMAIN macro by the task itself or by another task. No control blocks describe the gotten **common** storage.

The task-owned storage is described by two types of control blocks:

- Task storage headers (TSHs)
- Gotten storage blocks (GSBs).

As shown in Figure 11, the TSHs are blocked in blocks called task storage header blocks (TSHBs) and the GSBs are blocked in blocks called **blocks of gotten storage blocks** (GSBBs). The TSHBs are linked in a double linked list. Each TSH

points to a GSBB block (block of GSBs). Each GSB has the final description of a piece of gotten storage (address, length, subpool, and key).

The TSHB contains a block header followed by a string of TSHs. The GSBB contains a block header followed by a string of GSBs. Neither the TSHs in a TSHB nor the GSBs in a GSBB are linked together.

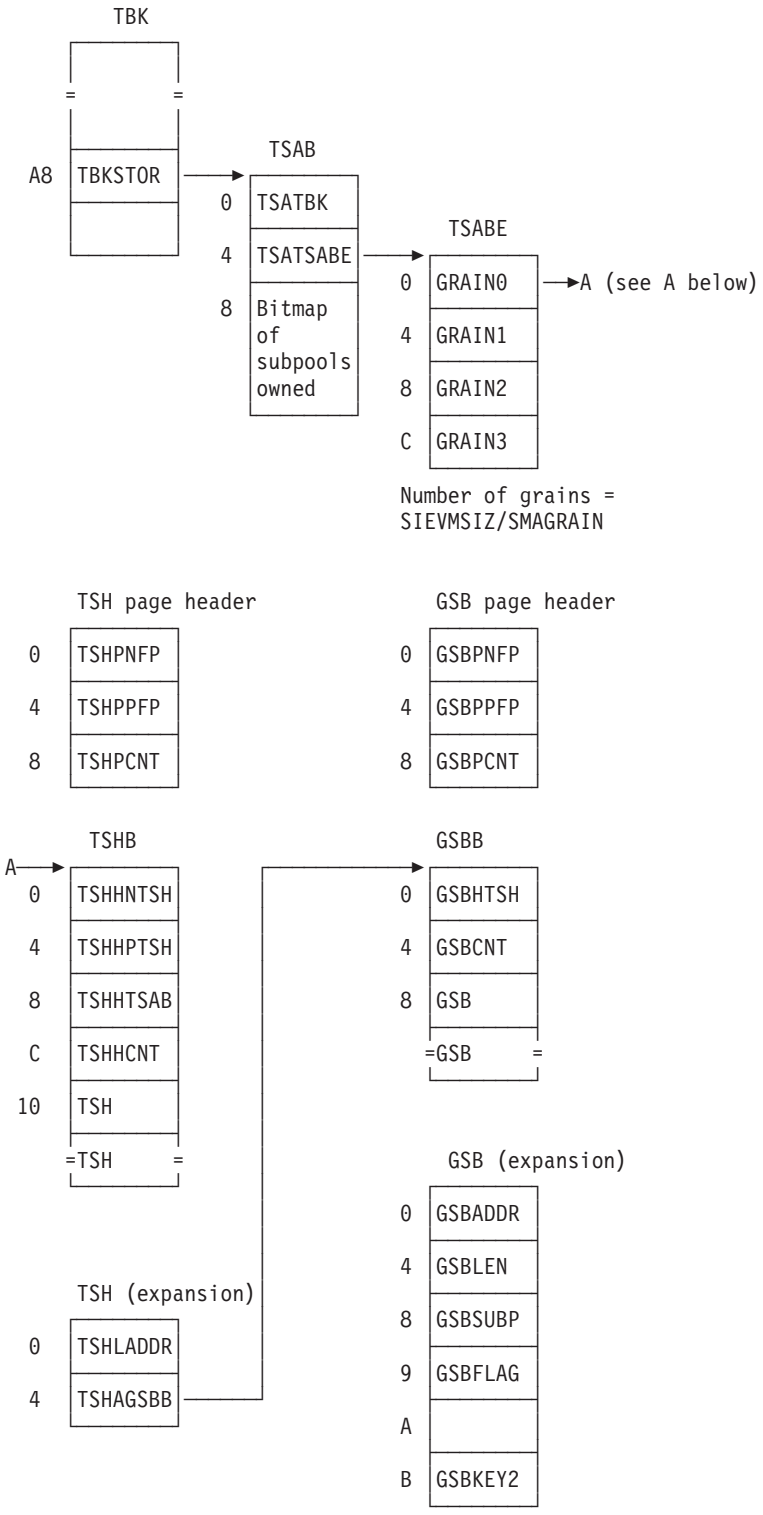


Figure 11. TSHB and GSBB Control Blocks

Each page of Task Storage Header (TSH) blocks contains a header at the beginning of the page. The fields in the page header are:

Disp	Label	Field Description
X'00'	TSHPNFP	Next page of TSH page blocks.

Disp	Label	Field Description
X'04'	TSHPPFP	Previous page of TSH page blocks.
X'08'	TSHPCNT	Number of used TSH blocks on this page.

The fields in the Task Storage Header Block (TSHB) are:

Disp	Label	Field Description
X'00'	TSHHNTSH	The link pointer to the next TSH block for the same task
X'04'	TSHHPTSH	The link pointer to the previous TSH block for the same task
X'08'	TSHHTSAB	The link pointer back to the TSAB
X'0C'	TSHHCNT	The number of TSHs in this block
X'10'		The first TSH in this block

The fields in a TSH are:

Disp	Label	Field Description
X'00'	TSHLADDR	The low address of the areas described by the GSBs in the corresponding GSB block
X'04'	TSHAGSBB	The address of the block of GSBs

The description of the relation between the TSHB and the block of TSHs is in the SMAB. The field descriptions are:

1. The length of a block of TSHs (including the block header) is in the SMATSMBL (SMAB + X'40').
2. The number of blocks of TSHs on a page is in SMATSHBN (SMAB + X'42').
3. The maximum number of TSHs in a block is in SMATSHBM (SMAB + X'44').

Each page of Gotten Storage Block (GSB) blocks contains a header at the beginning of the page. The fields in the page header are:

Disp	Label	Field Description
X'00'	GSBPNFP	Next page of GSBs.
X'04'	GSBPPFP	Previous page of GSBs.
X'08'	GSBPCNT	Number of used GSB blocks on this page.

The fields in a Block of Gotten Storage Blocks (GSBB) are:

Disp	Label	Field Description
X'00'	GSBHTSH	The link pointer back to the TSH
X'04'	GSBCNT	The number of GSBs in this block
X'08'		The first GSB in this block

The fields in a GSB are:

Disp	Label	Field Description
X'00'	GSBADDR	The address of the gotten storage
X'04'	GSBLEN	The length of the gotten storage
X'08'	GSBSUBP	The subpool of storage
X'09'	GSBFLAG	A flag byte containing, in the right-most bit, the flag showing whether the piece of storage described by the GSB is in key zero nonfetch-protected
X'0B'	GSBKEYZ	Key 0 non-fetch protected storage.

The description of the relation between the GSBB (block of GSBs) and the GSBs is also in the SMAB. Field descriptions are:

1. The length of a block of GSBs is in SMAGSBBL (SMAB + X'46').
2. The number of GSB blocks on a page is in SMAGSBBN (SMAB + X'48').
3. The maximum number of GSBs in a block is in SMAGSBBM (SMAB + X'4A').

How to Find the Storage Belonging to a Given Task

1. Find the task block (TBK) (see “Task Management” on page 140).
2. Find TBKSTOR (X'A8' into the TBK), which points to the task storage anchor block (TSAB).
3. TSATSABE (X'04' into the TSAB) points to the task storage anchor block extension (TSABE).
4. TSABTSHB (X'00' into the TSABE) points to the first TSHB (TSH block) of the array of TSHBs belonging to the task.

How to Check What Subpools Belong to a Given Task

1. Find the task block (TBK) (see “Task Management” on page 140).
2. TBKSTOR (X'A8' into the TBK) points to the TSAB.
3. TSASPOOL (X'08' into the TSAB) is a 256-bit map of all possible subpool values. Each subpool number that is owned by the task has the appropriate bit on. If the bit is off, then there is an owning task with the corresponding bit on. You can search up the task chain to find the owner of any given subpool by looking for the appropriate bit to be on. At least one task has the bit on. The commands task has all 256 bits on.

System-Wide Description of Storage

The total of your virtual machine size (including common storage, though not used as task-oriented storage) is divided into sections called **grains**. The size of a grain is determined at IPL time when the SMAB is built. The field describing the size of a grain is SMAGRAIN (SMAB + X'3C'). The field named SMATSBEL (SMAB + X'18') contains the number of existing grains times 4. Each grain has a pointer in the TSABE to the first TSHB for that task in that grain. Consequently, SMATSBEL represents the size of the TSABE (TSAB extension). Furthermore, there is, for each grain, a double-linked list of TSHBs pertaining to that task. The number of grains is fixed at IPL time; therefore, to find the anchor of TSHBs for a particular part of storage, you need to determine the pointer in the TSABE (a zero entry indicates there is no TSHB) pertaining to that particular grain.

System-Wide Description of TSHBs and GSBBs

The TSHBs and GSBBs reside on four (two for each type of block) double-linked lists of pages. All four are anchored in the SMAB.

For each type of block the two linked lists are:

- A list with full pages of TSHBs or GSBBs
- A list of pages containing space for at least one block (TSHB or GSBB).

The TSHBs are double-linked lists, and thus can reside on both lists of pages. The pointers in the TSABE anchor the lists of TSHBs for each grain and point somewhere on one of the two lists of pages to the first TSHB for that particular grain.

The anchors in the SMAB of the four double-linked lists of pages are the following:

Displacement	Field Description
X'1C'	Pointer to first page filled with TSHBs
X'20'	Dummy backward pointer
X'24'	A pointer to the first page of TSH blocks containing at least one free block
X'28'	A dummy backward pointer
X'2C'	A pointer to first page filled with GSB blocks
X'30'	A dummy backward pointer
X'34'	A pointer to the first page of GSB blocks containing at least one free block
X'38'	A dummy backward pointer

Each page from any of the four lists has a header, the blocks follow immediately afterward. The fields of the header are:

Displacement	Field Description
X'00'	A pointer to the next page of TSH or GSB blocks (TSHBs or GSBBs)
X'04'	A pointer to the previous page of TSH or GSB blocks (TSHBs or GSBBs)
X'08'	The number of used TSH or GSB blocks (TSHBs or GSBBs) on this page

Depending on the position of the page inside the list, the first or second position could be zero.

Common Storage Management Problems

FREEMAIN or GETMAIN goes into an infinite loop

1. GETMAIN or FREEMAIN is searching for the task that owns the subpool requested. The task chain or the TSABs may have been overlaid.
   a. This problem will show up on a task-related request.
   b. Find the active task and search the task chain for each ancestor task. See if any have been overlaid. (GETMAIN and FREEMAIN search back up the task chain to find the task that owns the subpool.)
   c. TBKSTOR (X'A8' into the task block) points to the task storage anchor block (TSAB).
   d. TSASPOOL (X'08' into the TSAB) is a 256-bit map of all the subpools owned by this task. Either the active task or one of the owning tasks must have the appropriate bit on for a given subpool. GETMAIN or FREEMAIN will continue to search until the owner of the subpool is found.

Abend 80A, 804, or 878: Improper length or insufficient virtual storage

1. Check the trace table for the length of the request. (Tracing is done for SVC invocations of GETMAINs and FREEMAINs. Branch entries to GETMAIN and FREEMAIN are not traced.).

If the length is valid, then check for fragmentation. (See “Checking for Storage Fragmentation” on page 156.)

2. If there is fragmentation, find out who has not freed the storage.
   a. Find out who is not freeing storage by first finding the key of the storage with the CP command DISPLAY K.
   b. If most of the storage allocated is in key 6, then VTAM is not freeing the storage.
   c. If most of the storage is in key 14, then storage is not being freed by an application such as RSCS.
   d. If most of the storage is allocated in key 0, the problem could be internal to GCS, or GCS could be getting storage in behalf of some application.
   e. Check both the allocated storage of the task blocks and the free storage described by the major/minor SACB for patterns. Are the same size pieces of free storage being left? All major SACBs are found in contiguous storage and can be easily scanned. All the minor SACBs that describe free storage can be found on pages of minor SACBs pointed to by ANCHPGMN found in the anchor blocks. Thus you can easily scan the minor SACBs.
   f. Check the trace table for the last GETMAINs. See if FREEMAINs are done for that storage.

Abend 778: One of the following could be true:

1. There is an invalid mode byte in SVC parameter list.
2. The program is returning storage in wrong key.
   a. It could be returning someone else’s storage.
   b. A privileged program could have changed the key.
3. Storage management ran out of storage for internal control blocks.

Check the following:

1. The parameter list set up by the macro.
2. Whether actual storage key matches what GCS storage management identifies as the key. For more information, see “Finding the Key for a Given Page” on page 157.
3. Fragmentation.

Tracing Storage Management

Supervisor tracing using ITRACE and ETRACE includes tracing GETMAINs and FREEMAINs (called through SVCs) as they occur in GCS. GETMAIN trace entries (X'08' type) and FREEMAIN trace entries (X'09' type) contain much of the same information:

- The task ID
- The storage address obtained or released
- The length of the storage
- The storage subpool
- The invoker’s address.

GETMAIN also includes the key of the storage being obtained.

General I/O

GCS General I/O (GENIO) Functions: All I/O except DASD and console I/O is performed using the GCS GENIO macro. However, because GCS does not provide any device specific code, using the GENIO macro requires that the application requesting the I/O has to perform all the related I/O control tasks, including error recovery.

You can use operands of the GCS GENIO macro to request the following functions:

OPEN is needed for an application to use and own a particular device. To open a device, the program provides the virtual device address and the address of an exit routine. GCS passes control to this exit routine whenever the opened device presents an I/O interrupt.

When a GENIO OPEN is issued, GCS gets a table entry for the GENIO table (GIOTB) for the device and initializes the entry.

A task or program may not open a device that is already open.

CLOSE
closes a device when the device is no longer needed.

GCS cleans up any I/O requests queued on the virtual channel queue, halts any active I/O, and deletes the entry from the GIOTB table. (See "The General I/O Table (GIOTB)" on page 166 for a discussion of the GIOTB table.)

The exit routine specified in the GENIO OPEN macro is no longer scheduled if I/O interrupts are received from the device.

MODIFY
modifies a CCW of an active I/O program. DIAGNOSE code X'28' is issued to CP to effect the CCW modification.

CHAR requests the characteristics (such as device class, type, and model) of a device.

GCS gets this information by using DIAGNOSE code X'210'.

The CHAR function does not require the device to be open in order to obtain the requested information.

START
starts an I/O operation to an open device.

For this operation, the program specifies the virtual device address and the address of a channel program to be run on the device. The channel program key is set to the PSW key of the program that issued the START.

GCS checks that:

- The device is open
- The device is not busy with another operation.

GCS issues a virtual SSCH instruction to the device.

GCS does not accept another START function to the device until the current operation completes. The end of the operation is identified by a device end interrupt.

STARTR
lets an authorized program use real channel programs with a dedicated device. Only real attached devices may use real channel programs.

If a device is not capable of real I/O (not a real device), a return code is set, and no further processing takes place.

The process of a STARTR function is similar to the START function, the only difference is GCS uses DIAGNOSE code X'98' instead of an SSCH instruction.

Note: A virtual machine must be authorized to issue DIAGNOSE code X'98'. This authorization is granted by specifying DIAG98 in the directory entry of the virtual machine (OPTION statement).

If the machine is not authorized for DIAGNOSE code X'98', a return code is passed to the program issuing the GENIO STARTR function. See the *z/VM: CP Planning and Administration* book for a description of the setup necessary to use DIAGNOSE code X'98'.

HALT forces GCS to halt the device.

General I/O in GCS lets a program drive any I/O device that is defined on the virtual machine except a DASD. Using the GENIO macro, a user can obtain, use, and release any I/O device. For further information on the GENIO Macro, see the *z/VM: Group Control System* book.

IOSAVE

Information pertaining to general I/O is found in the IOSAVE area. IOSAVE is used as a save area when I/O interrupts are being handled. It resides in private storage and is loaded during system initialization. The address of IOSAVE is found in the load map for the system. The user must have the load map (for the IOSAVE address) to do general I/O debugging for GCS.

IOSAVE gives an overall picture of general I/O in the GCS virtual machine at a point in time, such as the time of the dump:

- The I/O old PSW, containing the address of the interrupting device in the second halfword of the PSW
- The address of the first entry in the general I/O table linked list
- A pointer to the page fix table (PFXTB) that identifies the pages that have been locked in real storage
- The address of the last entry in the general I/O table related to GENIO processing (either from an I/O interrupt or from issuing a GENIO macro).

The IOSAVE block resides in private storage and is built during GCS initialization. The initial value of all fields in IOSAVE is 0.

To determine the start address of the IOSAVE control block, locate GCTIOSAV in the GCS nucleus map.

The IOSAVE contains the following information:

Displacement	Field Description
X'00'	A save area for registers (twice)
X'90'	The I/O old PSW
X'98'	The SCSW from the I/O causing the interrupt

X'A0'	A pointer to the general I/O table
X'A4'	The address of the page fix table
X'A8'	The address of the last entry (before the current) in the general I/O table
X'AC'	A real I/O authorization flag
X'B0'	The interrupt code
X'B2'	The instruction length
X'B4'	The address of the first entry in the subchannel identification table

The saved PSW and SCSW are stored in the IOSAVE from the last I/O interrupt.

The Subchannel ID Table (SIDTABLE)

At IPL time, a table is built containing the existing configuration. Each entry corresponds to one subchannel. This is the SIDTABLE, a linked list with an entry for every active device. The SIDTABLE is anchored in IOSAVE + X'B4', and it is cross-linked with the general I/O table. (Each entry in the general I/O table points to a SIDTABLE entry and the reverse if there is a correspondent entry in the general I/O table.)

A SIDTABLE entry provides information about the device as:

- Subchannel ID
- Subchannel address
- Virtual and real device characteristics
- The interrupt request block for the respective device
- The operation request block for the respective device.

The fields in a SIDTABLE entry are:

Displacement	**Field Description**
X'000'	The next subchannel pointer
X'004'	The subchannel ID
X'008'	The subchannel address
X'00C'	The virtual device type class
X'00D'	The virtual device type
X'00E'	The virtual device status
X'00F'	The virtual device flags
X'010'	The real device type class
X'011'	The real device type
X'012'	The real device model number
X'013'	The real device feature code
X'014'	The address of the GCTGIOTB entry
X'018'	The interrupt request block
X'118'	The operation request block

The General I/O Table (GIOTB)

The general I/O table (GIOTB) is found at IOSAVE + X'A0'. It is a linked list with an entry for every open device.

A GIOTB entry provides information about the device, such as:

- The device address
- The task ID and task block address of the task that has opened the device

Only one task can own a device at any one time. A task owns a device when it opens the device and loses ownership when it closes the device, or when the task ends.

- Several flags describing the status of the I/O activity on the device

  If the flag for "exit scheduled" is on, an asynchronous exit block (AEB), pointed to by GIOTB+X'38', contains information related to the exit and is enqueued on the AEB queue pointed to by the SIE at SIE+X'18'.
- Characteristics of the device (virtual and real)
- A pointer to the subchannel ID table (SID) correspondent entry.

The field IOSGIOTB is found at IOSAVE + X'A0'.

The general I/O table contains the following information:

Displacement	Field Description
X'00'	The address of the next entry in the table
X'04'	The device address
X'08'	The address of the task requesting an open
X'0C'	The task ID of the task requesting an open
X'0F'	Flags
	Byte **Field Description**
	1xxx xxxx I/O is active
	x1xx xxxx I/O is queued
	xx1x xxxx An asynchronous interrupt has been queued
	xxx1 xxxx An exit has been scheduled for asynchronous interrupt
	xxxx 1xxx An asynchronous interrupt has been queued
	xxxx x1xx An asynchronous interrupt is pending
	xxxx xx1x Wait
	xxxx xxx1 Format 1 type CCWs are being used
X'14'	The address of the exit when I/O has been completed (GIOEXIT)
	(1xxx xxxx) Call exit in AMODE 31
X'18'	The characteristics of the virtual device
X'1C'	The characteristics of the real device
X'24'	The address of the CCW to be started
X'38'	The address of the asynchronous exit block (AEB)
X'40'	The synchronous interrupt control block (ICB)
X'8C'	The asynchronous interrupt control block (ICB)
X'D8'	The address of SID table entry

I/O Interrupt Handling

The exit routine specified in the GENIO OPEN macro is provided with the SCSW from the interrupt, and with the sense bytes if a unit check occurred. When subsequent SCSWs are received, the status bytes are OR'd with the SCSW already stored in the interrupt control block.

The exit routine receives control in the key and state of the task that opened the device:

- If the task is an authorized program, the exit routine is entered with interrupts disabled.
- If the task is not an authorized program, the exit routine is entered with interrupts enabled.

Interrupt Control Blocks

Within each GIOTB entry are two interrupt control blocks (ICBs) that keep information about the last synchronous (GIOSICB) and asynchronous (GIOAICB) I/O interrupts for the device.

The asynchronous and synchronous ICBs are mapped alike, except that the synchronous ICB contains sense bytes in case of unit checks. The synchronous ICBs contain a 0 in the first byte, while the asynchronous ICBs contain a 1.

The ICBs contain the device address and the Subchannel Status Word (SCSW).

The interrupt control blocks contain the following information:

Displacement	Field Description
X'04'	The device address
X'08'	The first two words of the SCSW
X'10'	The sense bytes (synchronous only - 32 sense bytes)
X'40'	The complete SCSW from the interrupt

How to Find What Pages Are Locked by PGLOCK

The page fix table (PFT) keeps track of the virtual pages that are locked into real storage by the PGLOCK macro. When a page is locked, an entry for that page is added to the PFT. The entry is deleted from the PFT when the page is unlocked using the PGULOCK macro. The PFT entries are chained together and are pointed to from IOSAVE (IOSAVE + X'A4').

A PFT entry contains the following information:

Displacement	Field Description
X'00'	The address of the next PFT entry
X'04'	The virtual address of the page
X'08'	The real address of the page
X'0C'	The task ID that locked the page
X'0E'	Flag: **X'80'** AMODE 24 page

Finding Pages Not Paged in After a Page Fault

If you are using the pseudo page fault support by issuing the CP command SET PAGEX ON and the task block is waiting for page fault completion(s), you can find out what page it is by following these steps:

At X'13C' into the SIE, there is a pointer which points to a chain of ECBs (Event Control Blocks) that provide information about tasks waiting for a page of storage to be paged into real storage. Each ECB control block pointed to by the pointer at X'13C' into the SIE has the following format:

Displacement	Field Description

X'00'	The forward pointer to the next ECB control block
X'04'	The backward pointer to the previous ECB control block
X'08'	The address of the page having page fault
X'0C'	An ECB
	Byte 01 **Flag Field** **1xxxxxxx** The task is waiting for the page to be paged in
X'0D'	The three-byte address of state block of task waiting for page

Use the state block pointer to find the backward pointer to the task block that is waiting for the page to be paged in real storage.

The page fault address for the last page fault handled is at X'90' If the high order bit is on, GCS has been notified of the completion. The program interrupt code, which must be X'14' for a page fault, is at X'8E'

How to Find the Characteristics of a Device

The GENIO macro with the CHAR option gives information about a specific device. The data returned contains both real and virtual characteristics. The device does not have to be open for you to enter the GENIO CHAR macro.

If the device has been opened, an entry in the general I/O table (GIOTB) for that device has been made. The GIOTB contains both real and virtual characteristics for the device. If there is no real device associated with the virtual device, the real characteristics are zero.

I/O Debugging

I/O problems can occur in four areas: CP, GCS, VSCS, or VTAM and its applications. Indicators that there may be an I/O problem in one of these areas include:

- Printers or a SNA/CCS terminal that hang
- A VTAM link that does not initialize
- A questionable status returned from I/O.

When you suspect an I/O problem, you should first keep track of error messages and keep the console log, especially for VTAM. I/O problems generally require recreating the problem using traces. You can set traces for each area suspected of an I/O problem. Trace files are helpful to track the sequence of events following the handling of an I/O interrupt. Proceed as follows:

1. Set up traces for CP, GCS, VSCS, and VTAM by entering:

```
trsource id xx type gt user vtam
trsave for id xx on dasd
trsource enab id xx
vscs traceon (ext (starts the VSCS external trace)
etrace gtrace sio i/o group:
vtam f trace, id=luname, type=buf (or type=i/o) (starts the VTAM trace)
```

2. Recreate the problem.
3. Turn off the traces by entering:

```
trsource disa id xx
vtam f notrace, id=luname, type=buf (or type=i/o) (stops the VTAM trace)
etrace end (stops the GCS trace)
vscs traceoff (stops the VSCS trace)
```

If you want to do an internal trace:

1. Using ITRACE involves entering only the GCS and VTAM parts of this scenario:

   ```
   itrace gtrace (enables GCS to record GTRACE data in the internal trace table)
   vtam f trace,id=a01a3e0,type=io (instructs VTAM to record IO trace data)
   vtam f trace,id=a01a3e0,type=buf (instructs VTAM to record buffer trace data)
   ```

2. Recreate the problem.
3. Turn off the traces by entering:

   ```
   vtam f notrace,id=a01a3e0,type=io (stops the VTAM IO trace)
   vtam f notrace,id=a01a3e0,type=buf (stops the VTAM buffer trace)
   itrace gtrace off (stops GCS internal tracing for GTRACE).
   ```

Trace Table Entries

After tracing has completed, the trace events for all areas that were traced are found in the GCS internal trace table, unless a wraparound has occurred. If GCS is using an external trace, the trace entries are in the TRFILE created for the TRSOURCE trace ID. VTAM and VSCS entries in the trace table are entered as GTRACE entries.

GCS traces of I/O requests (type X'06') and interrupts (type X'03') contain information that may be useful when debugging I/O problems. For more information on debugging VTAM, see *VTAM Diagnosis Guide*.

Recreating the Problem

When unexpected results occur on terminals or other SNA devices, you should recreate the problem with VTAM and VSCS traces on. This helps isolate the failing component. Most hung LU conditions are not GCS problems; they are probably CP or VSCS problems.

Tracing I/O is important when trying to recreate an I/O problem. It is helpful to know the state and configuration of the system before and after I/O is processed.

When you track I/O for a VTAM application, you should look at the parameter list that is being passed to GCS in the GENIO macro:

- Set a trace stop at the beginning of the GCS GENIO module (GCTGIM). This address is found in the load map for GCS.
- When VTAM issues the GENIO macro for I/O processing, the trace will occur.
- Register 1 will point to the parameter list. Ensure that it is a valid parameter list.

Command and Console Support

The GCS VM operator uses the console to communicate with either the GCS supervisor or applications through commands. The GCS supervisor and the applications can communicate with the operator through write-to-operator (WTO) and write-to-operator-with-reply (WTOR) instructions.

Command and console support includes commands issued from a terminal by a user and commands issued through the CMDSI macro. A user can use the CMDSI

macro to enter GCS-, CP-, or LOADCMD-defined commands from within a program running in GCS. For more information on the CMDSI macro, see the *z/VM: Group Control System* book.

LOADCMD Command

The LOADCMD command is included in the command support. LOADCMD lets users define their own command names for an entry point within a module. The module must reside in a load library that the user has defined with the GLOBAL command.

When the command defined by LOADCMD is issued, the module containing the entry point gets control. For more information on LOADCMD, see the *z/VM: Group Control System* book.

The LOADCMD command uses the NUCEXT function to determine if a command is already loaded as a nucleus extension. If the nucleus extension does not exist, NUCEXT is used to establish a nucleus for the command.

The chain of NUCX blocks are pointed to by SIENUCX located in the SI extension at X'A4'.

The NUCX contains the following important fields:

Displacement	Field Description
X'00'	NUCXPRT points to the next NUCX block
X'04'	NUCXUWRD is the user fullword
X'08'	NUCXNAME names the command
X'10'	NUCXPSW points to the starting PSW for the nucleus extension
X'11'	NUCXKEY is the user's key-bit(8)
X'14'	NUCXENTR points to the entry point address
X'30'	NUCXADDR is the address of the NUCCBLK that corresponds to this entry point
X'34'	NUCXTASK contains the task ID of the establisher-fixed(16)

NUCON Information

NUCON has a command area that contains information about commands that have been issued. This area contains information such as the command input line, the tokenized parameter list, and the pointers to the extended argument list.

NUCON contains the following command areas:

Displacement	Field Description
X'2E8'	The command input line
X'388'	The tokenized parameter list
X'5B8'	The address of the command token
X'5BC'	The address of the beginning of the argument string
X'5C0'	The address of the end of the argument string
X'5C4'	The address of the SIE state descriptor block

The command input line contains the last command or commands the user entered from the terminal along with the tokenized parameter list. The tokenized parameter list is built in NUCON when the command and parameters are scanned and

validated. The extended parameter list is also built during the scanning, and the fields for the extended parameter list in NUCON are filled in. When issuing one or more commands from the command line, only the command token and parameter list of one of the commands are included in the extended parameter list.

SIE Information

The SIE state descriptor block contains a commands and console area. This area contains such information as ECBs, CCWs, and pointers to the queues for the commands, messages, and replies that have not yet been processed.

The SIE contains the following command and console areas:

Displacement	Field Description
X'54'	The attention interrupt ECB
X'58'	The I/O complete ECB
X'5C'	The output pending ECB
X'60'	The command ECB
X'64'	FLAGS
	1xxx xxxx Read I/O is in progress
	x1xx xxxx Write I/O is in progress
	xx1x xxxx An attention is pending
	xxx1 xxxx Output is pending
X'68'	The address of the first CMDBUF on the queue
X'6C'	The address of the last CMDBUF on the queue
X'70'	The address of the first WQE on the queue
X'74'	The address of the last WQE on the queue
X'78'	The address of the first ORE on the queue
X'7C'	The address of the last ORE on the queue
X'80'	The Read/Write CCW
X'88'	The No-Op CCW
X'90'	The ORE ID bits
X'9D'	The last assigned ORE ID

Each ECB in the SIE is 4 bytes long. The first byte in the ECB is the most important. If the first bit is set on, the ECB is waiting. If the second bit is on, the ECB has been posted.

The following queues are maintained by the communications task:

- CMDBUF
- Write queue elements (WQE)
- Operator reply elements (ORE).

Each of these queues is pointed to from within the SIE and contain elements that have not yet been processed. As a command, write message, or reply is processed, it is taken from the queue. The first element on each queue is the next element to be processed. The last element on each queue is the most recently added element to the queue.

The SIE contains two CCWs. The first CCW is used for READ/WRITE, the second CCW is a no-op. The CCW contains a command code (CC), a data address, and the length. The data address points to the data to be read or written. The length of the data is given in the length field.

A format 0 CCW is mapped as shown in Figure 12:

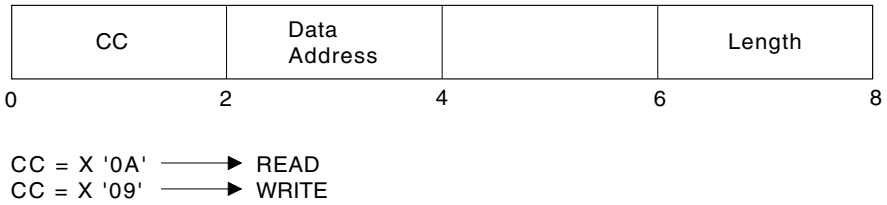


Figure 12. CCW Mapping

The ORE ID bits in the SIE are used to keep track of which reply numbers are outstanding (00 through 99). If the bit is on (1), the reply ID has been assigned, but the reply is still outstanding. When the ORE is built as a result of a WTOR instruction, the ORE ID is assigned from those that are available. When the reply is processed, the ORE is freed, and the ORE ID is made available again. (The bit associated with the ID is turned off.)

CMDBUF

The CMDBUF queue contains commands that have not yet been processed. Immediate commands are processed as soon as they are entered and are not entered into the CMDBUF queue. A CMDBUF element contains the command input data, the extended parameter list, and the tokenized parameter string. These fields correspond to fields in NUCON. The last CMDBUF in the queue contains the same information as in NUCON if it was the last command issued. If an immediate command was the last command issued, that command's parameter list is found in NUCON.

The CMDBUF element contains the following information:

Displacement	Field Description
X'00'	The next CMDBUF on the queue
X'04'	The length of the command data
X'08'	Command input data
X'8C'	The address of the command token
X'90'	The address of the start of the argument string
X'94'	The address of the end of the argument string
X'B0'	The tokenized parameter list

WQE and ORE

The WQE queue consists of messages to the VM operator. A WQE is built when a WTO or WTOR is issued. When the operator processes the WQE, it is taken from the queue. If a reply is expected (WTOR issued), a corresponding ORE is found in the ORE queue. The operator's reply is placed in the reply buffer pointed to by the ORE. If the message did not expect a reply (WTO issued), no corresponding ORE is present.

A WQE contains the following information:

Displacement	Field Description
X'00'	The address of the next WQE on the chain
X'06'	The length of the message text
X'08'	Message text

An ORE contains the following information:

Displacement	Field Description
X'00'	The address of the next ORE on chain
X'04'	The reply ID
X'08'	The address of the task block that issued the message
X'0C'	The length of the message text
X'10'	The message text
X'8C'	The key of the issuer
X'8D'	The length of the reply
X'90'	The address of the reply buffer
X'94'	The address of the reply ECB

A user can see if a message has not been processed by following the WQE chain, looking for a particular message. The end of the chain is reached when the next address in the chain is zero. If a WQE containing the message is not found, the message has been processed by the operator. If the message requested a reply, the user can follow the ORE chain, looking for the message and a reply. The user may also enter the QUERY REPLY command, which will return all messages that have outstanding replies.

VSAM

GCS supports a VSAM interface very similar to that supported by CMS. As in CMS, GCS supports an OS/MVS macro interface and maps these requests to VSE/VSAM. The VSAM operations are performed by the VSE/VSAM program.

Data Compression Services

The VSE/VSAM for VM Version 6 Release 1 (program number 5686-081) supports Data Compression Services to save DASD space in large customer databases. CMS and GCS will also support the VSE/VSAM for VM Version 6 Release 1 interface for Data Compression Services. When you use AMSERV to create a VSAM cluster, the COMPRESS parameter of the DEFINE function will allow record data to be compressed when it is written and will expand data when it is read. This parameter automatically lets VSAM know if the data is to be converted by VSAM when it is read or written; no application program changes are necessary.

Application Migration Considerations

An existing application can take advantage of these VSAM Data Compression Services without the need for program changes. The compression controls are in the VSAM product and are not tied to the application code. Two things must be done to migrate existing data sets to compressed format:

1. A 'VSAM.COMPRESS.CONTROL' KSDS compression control data set must be defined in each catalog where compressed data will reside.
2. The existing data set CLUSTER must be redefined as COMPRESS format.

Existing data sets can be unloaded temporarily so that the cluster can be redefined as compressed. The cluster can then be reloaded to create the compressed database which is immediately usable by application programs.

Data Compression Services will take advantage of the CMPSC hardware compression instruction, if available, to improve performance. Otherwise, a software simulation of the instruction will be used to execute the actual data compression.

Some return codes and feedback reason codes for Data Compression Services differ between MVS/VSAM and VSE/VSAM environments. For more detailed information on these differences, refer to the “OS/VSAM Error Codes” section of the *z/VM: CMS Application Development Guide for Assembler* for OPEN, CLOSE, and I/O Request error code tables.

GCS users can find error code information in the “VSAM Data Management Service Macros” section of the *z/VM: Group Control System* book.

For more information on VSE/VSAM Data Compression Services, see *VSE/VSAM Version 6 Release 1 Commands*, *VSE/VSAM Version 6 Release 1 User's Guide and Application Programming*, and *VSE/ESA Version 2 Release 1 Messages and Codes*.

Major differences between GCS and CMS for VSAM support include:

- AMS is not supported by GCS. Disk initialization, catalog definition, and file definition must be performed under CMS.
- All required VSE SVC simulation is part of the GCS nucleus. Therefore, there is no need to use a DOS segment.
- GCS includes basic support for VTAM.
- The SET SYSNAME command can only be used before the VSAM environment is initialized in GCS.
- GCS associates open ACBs with the task that performed the open. When a task completes, all open ACBs associated with that task are closed.
- Sharing of VSAM data in GCS is governed by VSAM and is the same as sharing VSAM data in a VSE partition.
- GCS supports Local Shared Resources (LSR) and Deferred Write (DFR) functions to enhance synchronous VM/VSAM processing.

This section concentrates on those areas in VSAM support that are unique to GCS or have changed from CMS. You should have some knowledge of how VSAM works in CMS and GCS, and the differences. More information on GCS support of VSAM is in the *z/VM: Group Control System* book. General information on VSE/VSAM support within VM is in the *z/VM: CMS Application Development Guide for Assembler*.

NUCON Changes

The GCS NUCON differs from the CMS NUCON in regard to VSAM support. The following is a summary of the changes in the NUCON for GCS support of VSAM and other information that is still found in the NUCON.

- The communications vector table (CVT) address is still located at X'10' in the NUCON. Neither the CMS nor GCS versions of the CVT table support all the fields defined in the MVS/OS environment. Only those fields used individually by the two VM subsystems are supported. However, the following are the two major differences between the CMS and GCS versions of the CVT:
  - The GCS version initializes its unsupported fields to X'0' values, while CMS initializes unsupported fields to X'FFFFFFFF' values.
  - The GCS version supported fields are a one-for-one match with MVS/OS supported fields as to the intent of the field definition. CMS supported fields may vary in some cases from the original intent of the MVS/OS definition.
- The VSE partition communications region (BGCOM) address, which is located at X'4E0' in the CMS NUCON, is located at X'14' in the GCS NUCON.

The following fields in the BGCOM have changed for GCS:

Displacement	Field Description
X'20'	The address of the VSAM anchor block minus 1
X'3B'	The dump option flag, which is always set
X'8C'	The flag for the GETVIS area initialized

The system communications region (SYSCOM) address, which is located at X'4E4' in the CMS NUCON, is at X'80' in the GCS NUCON.

The following fields in the SYSCOM have changed for GCS:

Displacement	Field Description
X'2F'	The XA hardware flag, which is now set

VAD Information

The VTAM/VSAM data block (VAD) supports VSAM on GCS. This data block resides in the first 64 KB segment of private storage in the GCS nucleus, the address of which can be found in the GCS nucleus load map. The VAD contains key addresses and other data relevant to running of VTAM and VSAM in GCS. This includes the addresses of the VSAM and BAM segments, the addresses of the VTAM OPEN, CLOSE, and CBMM routines, and pointers to the VSAM work areas chain, open ACBs list, and DOSCB chain.

The VAD contains the following information:

Displacement	Field Description
X'04'	The address of the first VSAM work area
X'08'	The address of the start of the VSAM segment
X'10'	The address of the start of the BAM segment
X'18'	The address of the first DOSCB
X'1C'	The addresses of the VTAM routines
X'28'	The address of the VSE transient area
X'30'	The address of the VSE lock table
X'34'	The address of the simulated VSE TCB
X'38'	The address of the VSE ppsave area
X'3C'	The address of the VSE LTA save area
X'40'	The number of DOSCBs in effect
X'88'	The address of the list of open ACBs
X'8C'	The length of the open ACBs list
X'90'	The address of VSAM VSRT table

Boundary Box Usage

The boundary box (BBOX), which normally shows the bounds of the partition in VSE, shows the bounds of a 16 MB virtual machine instead. Thus, all validity checks made by VSE/VSAM will be successful. GCS has its own address validation scheme, which is called before giving control to GCS/VSAM.

VSAM Anchor Block

In GCS, the anchor block contains the addresses of the VSAM dynamic assign table, VSAM AMCB table, VSAM OAL (OPEN ACB) table, Data Compression Services root block pointer, Data Compression Services gate word, and a reserved area for VSAM use. It does not contain the address of modules that are

CDLOADed, and it does not mark the boundary between GETVIS storage and partition storage, as CMS does. The VSAM anchor block is pointed to by the BGCOM.

VTAM/VSAM Work Areas

A VTAM/VSAM work area (VIPWORK) is established for each GCS task running VTAM/VSAM. The work areas are chained together with the newest task VIPWORK added to the beginning of the chain. VIPWORKs are removed from the chain when their related tasks end.

To find the VIPWORK:

- Locate the address of the VAD in the GCS nucleus load map
- Locate the address of the first VIPWORK at VAD + X'04'
- The address of the next VIPWORK is at VIPWORK + X'50'.

The VIPWORK contains the following information:

Displacement	Field Description
X'50'	The address of the next VIPWORK
X'54'	The address of the previous VIPWORK
X'58'	The address of the temporary OPEN/CLOSE ACB list
X'5C'	The size of the temporary OPEN/CLOSE ACB list
X'5E'	The task ID
X'7E'	Flags
	Byte — **Field Description** **1xxx xxxx** — PSW condition code = 0 **x1xx xxxx** — PSW condition code = 1 **xx1x xxxx** — PSW condition code = 2
X'80'	The save area for the caller's registers
X'BC'	The VIP entry caller return address
X'F0'	The DOS return code to the user

Helpful Hints for VSAM debugging

The following are GCS commands and macros you can use to get information about the state of the system at the current time.

QUERY SYSNAMES
: Displays the names of the standard saved systems or system names established through the SET SYSNAME command.

DLBL
: Without any operands specified, the current file definitions that were defined by the DLBL command are displayed.

SHOWCB
: A macro that returns the fields of a specified control block within VSAM.

TESTCB
: A macro that tests the values in the fields of a specified control block within VSAM.

IDUMP
: A VSAM IDUMP macro supported by GCS. GCS converts the request to an SDUMP macro for processing.

Debugging Data Compression Errors

After expanding a string of data, you may notice unexpected characters at the end of the string. To correct this, you must check the CMPSC_BITNUM bit in the CMPSC_DICTADDR_BYTE3 field of the CSRYCMPS area after a call to Data Compression Services. If this bit is on, you must add 1 to the length of the source area before calling Data Compression Services to expand your data. To test this bit, use a TM instruction.

Some return codes and feedback reason codes for Data Compression Services differ between MVS/VSAM and VSE/VSAM environments. For more detailed information on these differences, refer to the "OS/VSAM Error Codes" section of the *z/VM: CMS Application Development Guide for Assembler* for OPEN, CLOSE, and I/O Request error code tables.

GCS users can find error code information in the "VSAM Data Management Service Macros" section of the *z/VM: Group Control System*.

For more information on VSE/VSAM Data Compression Services, see *VSE/VSAM Version 6 Release 1 Commands*, *VSE/VSAM Version 6 Release 1 User's Guide and Application Programming*, and *VSE/ESA Version 2 Release 1 Messages and Codes*.

An Example of Control and Data Flow in GCS

The following is an example of the flow of a VTAM command that is entered by an application program. The diagram, shown in Figure 13, describes the configuration of a sample GCS group which contains five virtual machines:

- VTAM
- RSCS
- NetView
- An application (APPL)
- The recovery machine.

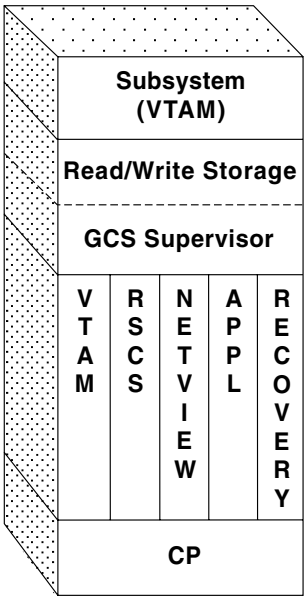


Figure 13. Sample GCS Group

A problem state application (APPL), running in its own virtual machine, issues the VTAM SEND macro. The VTAM SEND macro branches into an entry point in the VTAM shared segment. This entry point is filled in by VTAM when the application opened an ACB. The VTAM code, residing in the shared segment, issues the GCS AUTHCALL macro to enter another VTAM entry point in supervisor state. Now that the code is running in supervisor state, VTAM moves the data into common storage and issues a GCS SCHEDEX macro to signal the VTAM virtual machine in the group. The SCHEDEX function uses the CP signal system service to signal the VTAM virtual machine.

When CP dispatches the VTAM virtual machine, the GCS IUCV interrupt handler receives control to process the interrupt from the signal system service. The GCS IUCV interrupt handler passes control to a GCS module which schedules an asynchronous exit to run on a VTAM task, which may directly access the data in common storage. When that task is dispatched by the GCS dispatcher, it issues a GENIO STARTR to start the send on the virtual VTAM device. This must be done from the VTAM virtual machine because all VTAM GENIO devices are owned by the VTAM virtual machine. GENIO later receives a device end condition and schedules an I/O exit on the VTAM virtual machine, indicating the success of the operation.

Assuming the operation was successful and a response is required, the VTAM virtual machine receives an attention interrupt from the GENIO device. The VTAM virtual machine issues a GCS SCHEDEX to notify the application that issued the SEND of the response. SCHEDEX again uses the signal system service to schedule an exit (provided by VTAM) on the applications task that issued the SEND. The GCS Dispatcher then runs the VTAM exit on the applications task, and the exit informs the application through an interface provided by VTAM, completing the cycle for that SNA SEND.

Chapter 11. Debugging TSAF

The three ways that you can collect error information for problem diagnosis within Transparent Services Access Facility (TSAF) are described in this chapter. They are:

- Using console logs, described in “Using the Console Log” on page 182
- Using dumps, described in “Using TSAF Dumps to Diagnose Problems” on page 182
- Using system trace data, described in “Using System Trace Data to Diagnose Problems” on page 185.

In addition, “Interactive Service Queries” on page 187 describes how the TSAF QUERY command can also provide you with problem diagnosis information.

Note: The TSAF operator does not necessarily diagnose problems, especially from the TSAF virtual machine. Dumps and system trace data are usually used by a system programmer or whoever is responsible for diagnosing system problems.

Summary of Steps to Follow When a TSAF Abend Occurs

When a TSAF abend occurs, you should do the following:

1. Collect information about the error.
   - Save the console log or spooled console output from the TSAF virtual machine
   - Save and process any dumps that TSAF produces

     When an abend occurs in TSAF, either because TSAF issued an abend or because a TSAF or CMS operation caused a program exception, TSAF produces a dump through the CP VMDUMP command described in the *z/VM: CP Commands and Utilities Reference*. CP sends the dump to TSAF’s virtual reader.
   - Save any system TRFILE that contains TSAF data.
2. Collect other types of information about system status, such as:
   - Status of real and virtual devices that TSAF is using
   - System load at the time of the error on any systems using TSAF and the status of each system (for example, did another system abend?)
   - Types of applications that are using TSAF at the time and any information about them
   - Physical connection configuration of the systems in use.
3. Recover from the abend to continue processing.

   After TSAF creates a dump, it issues the LOAD PSW (LPSW) instruction. If TSAF is not invoked from the PROFILE EXEC, you must restart the TSAF virtual machine.

z/VM: Other Components Messages and Codes lists the TSAF abend codes and their causes.

Using the Console Log

TSAF provides informational messages, as well as error messages, that may help you with problem determination. To keep track of the console messages, enter:

```
spool console start to userid
```

where *userid* can be the user ID of the TSAF virtual machine or another virtual machine user ID to whom you want TSAF to send the console log. You may want to add this to TSAF's PROFILE EXEC so that a console log is always created.

To close the console log, enter:

```
spool console close
```

The log of messages received is sent to the specified user ID. See *z/VM: CP Commands and Utilities Reference* for more information on the SPOOL command.

TSAF provides additional information at the time of an abend to help you diagnose the problem. The console log contains information about the abend, such as:

- Abend code
- Program old PSW
- Contents of the general purpose registers.

TSAF also attempts to determine the displacement of the module in which the abend occurred and the displacement of the calling module.

Figure 14 shows some of the messages that TSAF may issue in response to an abend condition:

```
ATSCAC999T TSAF system error
ATSCAB017I Abend code ATS999 at 022730
ATSCAB018I Program old PSW is FFE002FF 40022730
GPR0-7 00022FFC 000003E7 00022FDA 00052BC0 00208080 00020C58 0033E811 00000001
GPR8-F 7F3B78AF 603C0000 00020B64 00022D6F 50021D70 00022B48 40022718 00023FB0
ATSCAB019I Abend modifier is ATSCAC
ATSCAB021I Failure at offset 0A06 in module ATSCAC dated 86.020
ATSCAB022I Called from offset 04B4 in module ATSSCN dated 86.078
ATSCAB023I VMDUMP ATSCAB*ATSCAB1 05/28/86 16:02:06 taken
```

Figure 14. Sample TSAF Console Log

Using TSAF Dumps to Diagnose Problems

You can use the Dump Viewing Facility to collect and diagnose problem data for the TSAF virtual machine. The console listing, as described in "Using the Console Log," may help you diagnose problems without using dumps.

These steps describe how to use dumps to diagnose TSAF problems:

1. Create a TSAF Dump Viewing Facility map, if it does not already exist
2. Create the TSAF dump
3. Process it
4. Diagnose it
5. Display it.

The sections that follow describe how to use the Dump Viewing Facility to perform this process.

Creating the TSAF Map

Note: You only need to do this step when a new CMS nucleus or TSAF module is built.

When a new CMS nucleus or TSAF module is built, enter the Dump Viewing Facility MAP command to compress the TSAF load map:

```
map cmsnuc map fm tsaf map fm (tsaf
```

The default names for the load maps are:

- TSAF MAP for the map source file
- CMSNUC MAP for the input CMS nucleus load map
- TSAFDVF MAP for the compressed map file, which you create using the MAP command.

Note: If you do not have the compressed map file, the power of the Dump Viewing Facility, which allows for diagnosis with dumps, is greatly reduced. For instance, without the map you cannot locate the TSAF modules by name.

For more information, see the *z/VM: Dump Viewing Facility* book.

Creating a TSAF Dump

The TSAF virtual machine creates its own dumps. The dump goes to the reader of the TSAF virtual machine. Because the TSAF virtual machine is not set up to process dumps, you need to transfer the dump file to the appropriate virtual machine.

If a dump of the TSAF virtual machine is necessary and the TSAF virtual machine did not abend, you can enter the VMDUMP command from the TSAF virtual machine console.

```
#cp vmdump 0-end system format tsaf
```

This CP VMDUMP command will dump the issuer's virtual storage contents from address 0 to the last address of storage and send it to the user ID designated as the dump receiver. This user ID is specified by the DUMP operand of the SYSTEM_USERIDS statement in the system configuration file. TSAF is the format type of the dump. The *z/VM: CP Commands and Utilities Reference* has more information about the VMDUMP command.

Processing a TSAF Dump

After the TSAF virtual machine creates a dump, load the dump onto disk. To load the dump, enter the following command:

```
dumpload
```

After you have loaded the dump onto a disk, append the map to the end of the dump by using the Dump Viewing Facility ADDMAP command:

```
addmap tsafdvf map a dumpname *
```

See the *z/VM: Dump Viewing Facility* book for more information on the ADDMAP command, and see the *z/VM: CP Commands and Utilities Reference* for more information about the DUMPLOAD utility.

Diagnosing a TSAF Dump

When you process a dump, a symptom record is generated. The symptom record helps you find out why TSAF created the dump. The symptom record includes:

- Information about the system environment at the time of the dump
- The symptom string that contains the following component-related symptoms:
  - The error code
  - The ID of the failing component
  - The ID of the failing module
  - The register and PSW contents.

When you use the Dump Viewing Facility DUMPSCAN command, the TSAF symptom record extraction routine updates the symptom record. You can use a version of the TRACE subcommand, provided specifically for TSAF, to format TSAF trace entries.

Note: TRACE is normally available only for CP dumps.

Displaying the TSAF Dump Information

The FDISPLAY subcommand of the DUMPSCAN command displays data control blocks, tables, and arrays important to the TSAF virtual machine. You can get information about the following by invoking different FDISPLAY parameters:

- Path array (PATH)
- Service table (SERVICE)
- Collection control block (COLLECT)
- Resource table (RESOURCE)
- Neighbor table (NEIGHBOR)
- Routing array (ROUTING)
- Link definition array (LINKDEF)
- Link control blocks (LINKCTL with types APPC, BSC, CTCA, ELAN, TLAN).

See the *z/VM: Dump Viewing Facility* book for a complete listing of FDISPLAY parameters.

Formatting and Displaying Trace Records in a Dump

TSAF maintains an internal trace table within the TSAF virtual machine. You can use the TRACE subcommand of DUMPSCAN to format and display trace records from the TSAF internal trace table. By using the HEX or FORMAT parameters, you can display the trace table entries in a hexadecimal display or a formatted display.

You can scroll back and forth through the formatted or hexadecimal output by using the DUMPSCAN subcommands FORWARD and BACKWARD.

Printing a TSAF Dump

If you want a listing of the dump, you can print one. The Dump Viewing Facility PRTDUMP command prints the dump and symptom record that DUMPLOAD processed. The output you get consists of the following:

- A symptom record
- A dump in hexadecimal (no special formatting)
- Appended load maps
- Contents of the registers and the PSW.

See the *z/VM: Dump Viewing Facility* book for more information on the PRTDUMP command.

Because of the recommended size of the TSAF virtual machine, the dump could be quite large.

Using System Trace Data to Diagnose Problems

While maintaining an internal trace table, the TSAF virtual machine can write trace entries to the system TRFILE. You can use the Dump Viewing Facility to format and display these trace table entries.

Setting External Tracing

The TRSAVE command specifies where you want to save the data. The TRSOURCE command controls the collection of the TSAF information. This information helps with problem determination. The TSAF SET ETRACE command lets you enable or disable external tracing for the TSAF virtual machine. You can trace data on specific links to the TSAF virtual machine. You can also trace data for other virtual machines (user IDs) that have established an APPC/VM path through the TSAF virtual machine.

To be sure that all trace data is recorded, enter the TRSOURCE command before issuing the SET ETRACE command. The users who enter the TRSOURCE command must have a Class C privilege user ID. In many locations, the TSAF virtual machine does not have the privilege class to issue the TRSOURCE command. For this reason, you may need to enter the command from another virtual machine that has authority to do so. [7]

To activate TRSOURCE for TSAF records only and to pass blocks of trace data to CP, enter:

```
trsource id tsafid type gt block for user userid
trsave for id tsafid
trsource enable id tsafid
```

tsafid is the trace identifier, and *userid* is the TSAF virtual machine user ID.

To activate TRSOURCE for TSAF records only and to pass individual records to CP, enter:

```
trsource id tsafid type gt event for user userid
trsave for id tsafid
trsource enable id tsafid
```

After you have entered the TRSOURCE command, you can begin to collect TSAF trace records. Enter the following from the TSAF virtual machine console:

```
set etrace on
```

When you set external tracing on, certain internal TSAF trace records are written externally to a system trace file (TRFILE). A complete description of the SET ETRACE command is in the*z/VM: Connectivity* book.

To end TSAF trace record generation, enter:

```
set etrace off
```

7. Privilege class is defined in the directory entry for the user ID.

To end TRSOURCE processing, enter:

trsource disable id *tsafid*

When you enter this command, the output data is stored as a system trace file (TRFILE).

To delete the trace ID, enter:

trsource drop id *tsafid*

For more specific information about the TRSOURCE and TRSAVE commands, see the*z/VM: CP Commands and Utilities Reference* book.

Viewing TSAF Trace Entries

You can use the CP TRACERED utility to format and print or view the trace entries. The DUMPSCAN command displays the external trace entries. In order to use the TRACE subcommand, the TSAF trace formatting routines must be on an accessed disk.

For information about the TRACERED utility, see the *z/VM: CP Commands and Utilities Reference*.

For information about the DUMPSCAN command and the TRACE subcommand, see the *z/VM: Dump Viewing Facility* book.

Trace Table Entry Format for TSAF

The trace table entries vary in length and follow the format described below. The length fields are 2 bytes long and may be any number from 0 to 32767. The length and data fields are optional data fields.

A trace table entry looks like the following:

length(1)	data(1)	...	length(n)	data(n)	Trailer record

The trailer record format looks like the following:

Clock (STCK format)	Characters 4 through 6 of module name	Trace ID code	Data area length	X'E00E'

The lengths associated with these fields are:

- Clock (STCK format)—8 bytes
- Characters 4 through 6 of module name—3 bytes
- Trace ID code—2 bytes
- Data area length—2 bytes
- ‘E00E’x—2 bytes.

Note: Module entries and module exits do not have length fields associated with each data field. Module entries and exits do, however, have the data area length in the trailer record.

Module entry trace records appear only in the internal trace table. TSAF identifies these records by setting bit 15 of the trace identifier code to 1. The data for a module entry is in the parameter list used during the module call.

Module exit trace records also appear only in the internal trace table. TSAF identifies these records by setting bit 14 of the trace identifier code to 1. The data for a module exit is in registers 14 and 15 at the time of the module exit.

Interactive Service Queries

The TSAF QUERY command, issued from the TSAF virtual machine, can give you more information to help you diagnose problems. The TSAF QUERY command gives you data about the TSAF configuration when the TSAF virtual machine is running:

- QUERY COLLECT displays the processor names that are currently in the TSAF collection.
- QUERY ETRACE displays the current setting of the external tracing.
- QUERY GATEWAY displays the current list of gateways defined in the TSAF collection.
- QUERY LINK displays information about the links that TSAF currently has including the neighboring processor name that the link is connected to.
- QUERY RESOURCE displays the current list of global resources in the collection.
- QUERY ROUTE displays the route information at the node where the command was issued.
- QUERY STATUS displays the current information about the correlation of other TSAF virtual machines in the collection.

See the *z/VM: Connectivity* book for more specific information about the TSAF QUERY command.

Chapter 12. Debugging AVS

Effective problem diagnosis for APPC/VM VTAM Support (AVS) is a process consisting of:

- Analyzing the dump
- Analyzing system trace data
- Using the AVS QUERY command
- Receiving an AVS abend response.

Each of the above steps will be addressed individually.

Note: The AVS operator does not always diagnose problems. In fact, dumps and system trace data are often handled by a system programmer or other person specifically responsible for diagnosing system problems.

Using AVS Dumps to Diagnose Problems

The Dump Viewing Facility analyzes dumps and tracks problems in z/VM. You can use the Dump Viewing Facility to collect and diagnose problem data for the AVS virtual machine. Because AVS runs in a GCS group, you can use all GCS and AVS subcommands of DUMPSCAN.

The steps used to diagnose problems using dumps are:

1. Obtain a GCS load map, if one doesn't already exist
2. Obtain the AVS dump
3. Process the AVS dump
4. Use DUMPSCAN to diagnose the AVS dump.

Obtaining the GCS Load Map

Note: This step is not necessary every time you create a dump; however, it is **required** when a new GCS nucleus is built.

When you build a new GCS nucleus, enter the MAP command of the Dump Viewing Facility to compress the GCS load map into a format that the Dump Viewing Facility can use:

```
map gcsnuc map fm (gcs
```

The default map file is GCSDVF MAP. See the *z/VM: Dump Viewing Facility* book for more information on the MAP command.

If you do not have the GCS load map, the GCS subcommands for the DUMPSCAN command are affected. The AVS subcommands for the DUMPSCAN command are unaffected.

Creating an AVS Dump

When a problem occurs because of an abend, or when an abnormal condition is detected, AVS produces one of the following:

- A dump when AVS abends
- A problem dump when the system detects an error but does not cause AVS to abend.

The problem dump takes a snapshot of the system to try to capture the problem. An informational message appearing at the operator console corresponds to a message number generated with the problem dump report. DIAGNOSE code X'94' (VMDUMP) is used to take the problem dump.

The maximum number of AVS problem dumps that can be taken during each AVS session is determined by the value set for MAXPROBD in the AGWTUN ASSEMBLE file. The default is 20. This, and other IBM-supplied default values contained in AGWTUN ASSEMBLE can be changed by a system programmer. For information about modifying this file, see the *z/VM: Connectivity* book.

If you want to create a dump for the AVS machine, enter:

```
gdump 0-end format avs dss
```

This Group Control System GDUMP command will dump the issuer's virtual storage contents, from address 0 to the last address of virtual storage, and send it to the issuer's virtual reader. AVS is the format type of the dump. The dump will also include any discontiguous saved segments that the virtual machine may be using. The *z/VM: Group Control System* book contains more information about the GDUMP command.

Processing an AVS Dump

To load any AVS virtual machine dump directly onto a disk, enter:

```
dumpload
```

After you have loaded the dump onto the disk, append the map to the end of the dump by using the Dump Viewing Facility ADDMAP command:

addmap gcsdvf map a *dumpname* *****

See the *z/VM: Dump Viewing Facility* book for more information on the ADDMAP command, and see the *z/VM: CP Commands and Utilities Reference* for more information about the DUMPLOAD utility.

Diagnosing an AVS Dump

When you process a dump, a symptom record is generated. The symptom record helps you discover why AVS created the dump. The symptom record includes:

- Information about the system environment at the time of the dump
- The symptom string that contains the following component-related symptoms:
  - The error code
  - The ID of the failing component
  - The ID of the failing module
  - Register and PSW contents.

When you use the Dump Viewing Facility DUMPSCAN command, the AVS symptom record extraction routine updates the symptom record. You can use a version of the TRACE subcommand, provided specifically for AVS, to format AVS trace entries.

Displaying the AVS Dump Information with DUMPSCAN

The GDISPLAY subcommand of DUMPSCAN displays data control blocks and addresses important to the AVS virtual machine. You can get information about the following by invoking different GDISPLAY parameters:

- Conversation block (CVB)

- Global control block (GCB)
- Gateway block (GWB)
- Gateway parameters (GWBPTRS)
- Module names (MAPA)
- Module addresses (MAPN)
- Remote LU block (RLU)
- Subtask control block (SCB)
- Scheduling global block (SGB).

Because AVS runs in a GCS group, you can use other DUMPSCAN subcommands to further examine these parts of the AVS dump:

IUCV All entries in the IUCV path table.

TACTIVE The task's active program list.

TLOADL The task's load list.

TSAB The subpool map and chain header of a task.

VMLOADL Information about all programs loaded in this virtual machine.

See the *z/VM: Dump Viewing Facility* book for more information.

Formatting and Displaying Trace Records in a Dump

AVS maintains an internal trace table within the AVS virtual machine. You can use the TRACE subcommand of DUMPSCAN to format and display trace records from the AVS internal trace table. By using the HEX or FORMAT parameters, you can display the trace table entries in a hexadecimal display or a formatted display. See the *z/VM: Dump Viewing Facility* book for examples of using the TRACE subcommand and the sample outputs.

You can scroll back and forth through the formatted or hexadecimal output by using the DUMPSCAN subcommands FORWARD and BACKWARD.

Using System Trace Data to Diagnose Problems

While maintaining an internal trace table, the AVS virtual machine can also write trace entries to the system trace file (TRFILE). You can use the Dump Viewing Facility to format and display these trace table entries.

Setting Internal Tracing

When the AGW START command is entered, internal tracing is set as if you entered an AGW SET ITRACE ALL ON command. Internal tracing information is written to an internal wraparound table in the AVS virtual machine.

See the description of the AGW SET ITRACE command is in the *z/VM: Connectivity* book for information about tracing events for a gateway or for stopping and restarting tracing.

Setting External Tracing

The TRSAVE command specifies where you want to save trace information. The TRSOURCE command controls the collection of the data. This information helps with problem determination. The AGW SET ETRACE command lets you enable or disable external tracing for the AVS virtual machine. External tracing will not be in effect unless you also have internal tracing set on. The type of external tracing you

receive will be the same as the type of internal tracing you requested. To be sure that all trace data is recorded, enter the TRSOURCE command before issuing the AGW SET ETRACE command. The users who enter the TRSOURCE command must have a class C privilege user ID. Because the AVS virtual machine is not set up to diagnose problems, only one authorized user at a time may enter the TRSOURCE command. [8]

To activate TRSOURCE for AVS records only and to pass blocks of trace data to CP, enter:

```
trsource id avsid type gt block for user userid
trsource enable id avsid
```

avsid is the trace identifier, and *userid* is the AVS virtual machine user ID.

To activate TRSOURCE for AVS records only and to pass individual records to CP, enter:

```
trsource id avsid type gt event for user userid
trsource enable id avsid
```

After you have entered the TRSOURCE command, you can begin to collect AVS trace records. Enter the following from the AVS virtual machine:

```
etrace gtrace
agw set etrace on
```

When you have internal and external tracing set on, AVS trace records are written externally to a system trace file. The ETRACE command is described in the *z/VM: Group Control System* book. A complete description of the AVS SET ETRACE command is in the *z/VM: Connectivity* book.

To end TRSOURCE processing, enter:

```
trsource disable id avsid
```

When you enter this command, the output data is stored as a system trace file (TRFILE). For more specific information about the TRSOURCE command, see the *z/VM: CP Commands and Utilities Reference*.

Viewing AVS Trace Entries

You can use the CP TRACERED utility to format and print or view the trace entries. The DUMPSCAN command also displays the external trace entries. In order to use the TRACE subcommand, the AVS trace formatting routines must be on an accessed disk.

For information about the DUMPSCAN command and the TRACE subcommand, see the *z/VM: Dump Viewing Facility* book.

For information about the TRACERED utility, see the *z/VM: CP Commands and Utilities Reference*.

Trace Table Entry Format for AVS

AVS trace table entries vary in length and follow the format described below. The length fields are 1 byte long and may contain any number from 0 to 236. An AVS trace entry cannot exceed 255 bytes. The length and data fields are optional. A trace entry table looks like the following:

8. The privilege class is defined in the directory entry for the user ID.

length(1)	data(1)	...	length(n)	data(n)	Trailer record

The trailer record format looks like the following:

Clock (STCK format)	Characters 4 through 6 of module name	Trace ID code	Data area length	X'E00E'

The lengths associated with these fields are:

- The clock (STCK format)—8 bytes
- Characters 4 through 6 of the module name—3 bytes
- The trace ID code—2 bytes
- The data area length—2 bytes
- X'E00E'—2 bytes.

Getting Information about Trace Entries

You can use the CP QUERY command to obtain information about traces and trace entries. For example,

- QUERY TRFILES displays detailed information about one or more system trace files and counts the number of files that match your specified criteria.
- QUERY TRSAVE displays the destination of the traces.
- QUERY TRSOURCE displays the status of traces defined by TRSOURCE.

For information about the QUERY command, see the *z/VM: CP Commands and Utilities Reference*.

Interactive Service Queries

The AVS QUERY command provides information about the operating AVS virtual machine.

- AGW QUERY ALL displays all of the current information about various settings and conditions of AVS.
- AGW QUERY CNOS displays the contention winner/contention loser information for the gateways.
- AGW QUERY CONV displays information about the current conversations.
- AGW QUERY ETRACE displays the current setting of the external tracing.
- AGW QUERY GATEWAY displays the status of all gateways that are currently in the collection.
- AGW QUERY ITRACE displays the current setting of the internal tracing.
- AGW QUERY USERID displays the remote LU, remote user ID, and local user ID.

See the *z/VM: Connectivity* book for more information about this command.

Summary of Steps to Follow When an AVS Abend Occurs

When an AVS abend occurs, follow these procedures:

- Collect information about the error.
  - Print the console log for the time that the error occurred. Save the console sheet or spooled console output from the AVS virtual machine.

  - Save and process any dumps that AVS produces.
  - Enter the MAP command to convert the GCS load map to a format that allows the Dump Viewing Facility to append the GCS load map to the dump.
  - Use the DUMPLOAD utility to load the dump from a reader spool file into a CMS dump file.
  - Enter the ADDMAP command to append the load map to the dump.
  - Enter the DUMPSCAN command with the necessary subcommands to look at the contents of the dump.
  - Save any trace files that contains AVS data (described in “Using System Trace Data to Diagnose Problems” on page 191).
- Collect system status information. The following information can help better determine problems:
  - The system load at the time of failure on any systems using AVS and the status of each system (for example, did another system abend?).
  - The types of applications that are using AVS at the time, and any information about them.
  - The physical connection configuration of the systems in use.
- Recover from the abend to continue processing.
  - When an abend occurs in AVS, either because AVS issued an ABEND or because an AVS or GCS operation caused a program exception, AVS produces a dump by way of DIAGNOSE code X'94' (described in *z/VM: CP Programming Services*).

z/VM: Other Components Messages and Codes lists the various AVS abend codes and their causes.

Appendix A. Problem-Specific Checklists

After you determine the general nature of your problem, find the checklist associated with that problem. Then, collect the information stated in the checklist before you call IBM.

CP Abend Checklist

Collect the following information before calling IBM:

1. The last action performed by CP before the abend occurred
2. Any output generated that demonstrates the problem
3. Any messages and return codes received
4. A CP restart or snapdump
5. A CP abend dump
6. A CP nucleus loadmap
7. If possible, the program label or the address at which the abend occurred.

CMS Abend Checklist

Collect the following information before calling IBM:

1. The last action performed by CMS before the abend occurred
2. Any output generated that demonstrates the problem
3. Any messages and return codes received
4. At a minimum, the contents of the PSW and the general and control registers
5. A dump of the virtual machine containing CMS
6. A CMS nucleus loadmap
7. If possible, the program label or the address at which the abend occurred.

GCS Abend Checklist

Collect the following information before calling IBM:

1. The identity of the virtual machine in the GCS virtual machine group that experienced the abend
2. A dump of the virtual machine that terminated abnormally
3. Any output generated that demonstrates the problem
4. Any messages and return codes received
5. A GCS nucleus loadmap
6. If possible, the program label or the address at which the abend occurred.

RSCS Abend Checklist

Collect the following information before calling IBM:

1. The last action performed before the abend in RSCS occurred
2. Any messages and return codes received
3. The RSCS console log
4. An RSCS abend dump
5. The RSCS nucleus loadmap (RSCS Version 1)
6. The RSCS link edit map (RSCS Version 2 or higher)

7. If possible, the program label or the address at which the abend occurred.

CP Wait State Checklist

Collect the following information before calling IBM:

1. The last action performed by CP before the wait state occurred
2. A CP restart or standalone dump
3. Any output generated that demonstrates the problem
4. The contents of the PSW. (Take particular note of the last word of the PSW. A CP wait state code might be stored there.)
5. The contents of the general registers
6. A copy of the CP internal trace table. (This accompanies the dump.)
7. If available, the wait state code.

Virtual Machine Wait State Checklist

Collect the following information before calling IBM:

1. The last action performed by the virtual machine in question
2. Any output generated that demonstrates the problem
3. Any messages and return codes received
4. The contents of the PSW
5. The contents of the general and control registers
6. The contents of the CSW. (Take particular note of CSW bits 32 through 47 where input/output device conditions might be noted.)
7. A dump of the virtual machine in question
8. If available, the wait state code.

RSCS Wait State Checklist

Collect the following information before calling IBM:

1. The last action performed by the virtual machine in question
2. Any output generated that demonstrates the problem
3. Any messages and return codes received
4. The contents of the PSW
5. The contents of the general and control registers
6. The contents of the CSW. (Take particular note of CSW bits 32 through 47 where input/output device conditions might be noted.)
7. A dump of the RSCS virtual machine
8. The RSCS console log
9. The RSCS nucleus loadmap (RSCS Version 1)
10. The RSCS link edit map (RSCS Version 2 or higher)
11. If available, the wait state code.

Application Program checklist for Unexpected Output

Collect the following information before calling IBM:

1. Documentation associated with the application program
2. Input to the program
3. The job control statements (JCL) included with the program.

Checklists for Performance Problems

An Infinite Loop in CP

Collect the following information before calling IBM:

1. Any console or printed output that demonstrate the problem
2. A CP restart dump
3. The contents of the PSW
4. The contents of the general and control registers
5. The contents of storage locations from hexadecimal addresses 00 through 100
6. If possible, the instructions (and their addresses) that are involved in the loop
7. A CP nucleus loadmap—particularly the names of the modules involved in the loop.

An Infinite Loop in a Virtual Machine

Collect the following information before calling IBM:

1. Any output generated that demonstrates the problem
2. A dump of the virtual machine in question
3. A CMS nucleus loadmap
4. If possible, the instructions (and their addresses) that are involved in the loop.

An Infinite Loop in RSCS

Collect the following information before calling IBM:

1. Any output generated that demonstrates the problem
2. The RSCS nucleus loadmap (RSCS Version 1)
3. The RSCS link edit map (RSCS Version 2 or higher)
4. The RSCS console log
5. A trace of activity in the RSCS virtual machine
6. If possible, the name of the RSCS module involved
7. If possible and if applicable, the name of the RSCS link or line driver involved.

Hardware Failure

Collect the following information before calling IBM:

1. Any messages and return codes received
2. The hardware error record.

Inadequate System Parameters

Collect the following information before calling IBM:

1. Normal system parameter readings
2. Present system parameter readings

3. The configuration of your system's input/output devices.

Appendix B. GCS Control Blocks

This appendix describes the layouts of some GCS control blocks and important fields that help you identify problems in a VM/SNA environment. The information that is provided is enough to allow you to display the GCS areas that may be relevant when determining the source of a problem.

This appendix describes the format and layout of:

NUCON The GCS nucleus constant area (Table 5)

SIE The NUCON extension (Table 6)

TBK The task block (Table 7)

STBLK The state block (Table 8)

SMAB The storage management block (Table 9)

ANCH The storage anchor block (Table 10)

EXTWA The external interrupt handler work area (Table 11)

SVCWA The SVC interrupt handler work area (Table 12)

PGMWA The program interrupt work area. (Table 13)

VMCB The virtual machine control block (Table 14)

In all the descriptions, the field lengths are shown in hexadecimal.

NUCON—GCS Nucleus Constant Area

Table 5. Contents of the GCS Nucleus Constant Area (NUCON)

HEX DISP	NAME	LENGTH	DESCRIPTION
000	NUCON	1880	The nucleus constant area
000	NUCIPPSW	8	The initial program loading PSW
000	NUCRNPSW	8	The RESTART new PSW
008	NUCROPSW	8	The RESTART old PSW
010	NUCADCVT	4	The address of the OS CVT
014	NUCBGCOM	4	The address of BGCOM
018	NUCEOPSW	8	The external old PSW
020	NUCSOPSW	8	The SVC old PSW
022	NUCSOBT2	1	Byte 2 of PSW
	NUCSOAS1	X'80'	The first address space control bit
	NUCSOAS2	X'40'	The second address space control bit
024	NUCSOADR	4	The XA SVC instruction address
	NUCSOA31	X'80'	The AMODE SVC old PSW
028	NUCPOPSW	8	The program-check old PSW
030	NUCMOPSW	8	The machine-check old PSW
038	NUCIOPSW	8	The I/O old PSW
04C	NUCACVT2	4	The CVT address for dump routines
054	NUCTRACE	4	The address of the table trace header
058	NUCENPSW	8	The external new PSW
060	NUCSNPSW	8	The SVC new PSW
068	NUCPNPSW	8	The program-check new PSW
070	NUCMNPSW	8	The machine-check new PSW
078	NUCINPSW	8	The I/O new PSW

Table 5. Contents of the GCS Nucleus Constant Area (NUCON) (continued)

HEX DISP	NAME	LENGTH	DESCRIPTION
080	NUCSYSCM	4	Used by VSAM
084		2	Reserved—set to zero
086	NUCEICOD	2	The external interruption code
088		1	Reserved—set to zero
089	NUCSVILC	1	The SVC ILC (XA and XC virtual machine)
08A	NUCSVCN	2	The SVC interruption code (XA or XC virtual machine
08C		1	Reserved—set to zero
08D	NUCPIILC	1	The program-check ILC
	NUCPILC1	X'04'	The program instruction length bit 1
	NUCPILC2	X'02'	The program instruction length bit 2
08E	NUCPICOD	2	The program interruption code
090	NUCTE	4	The page fault address
090	NUCTEA	1	Reserved—set to zero
	NUCTEAC	X'80'	The page fault is complete
091	NUCTEAA	3	The translation exception address
094		1	Reserved—set to zero
095	NUCMCNUM	1	The Monitor CALL class number
096	NUCPERCD	1	The program event recorder code
097		1	Reserved—set to zero
098	NUCPER	1	Reserved—set to zero
099	NUCPERAD	3	The program event recorder address
09C	NUCEID	4	The MONITOR-CALL EID
09D	NUCMTRCD	3	The MONITOR-CALL code
0A0	NUCEXAID	1	The exception access ID
0A8	NUCMCKLA	8	The machine-check LOGOUT area
0A8	NUCTXCP	4	The exception alet
0A8	NUCCHNID	4	The channel ID
0AC	NUCIOEL	1	Reserved for future use
AD	NUCIOELA	3	The I/O extended LOGOUT pointer
0B0	NUCLCL	4	The limited channel LOGOUT (ECSW)
0B8	NUCIOSID	4	The SID causing I/O interrupt
0B8	NUCIOSTY	2	The SID type
0BA	NUCIOAA	2	The I/O device causing interrupt The I/O subchannel causing INTR
0BC	NUCINTP	4	The interrupt parameter
0C0	LOWSAVE	96	The save area for the first 96 bytes of storage
0E8	NUCMCIC	8	The machine check interrupt code
0E8	NUCMCIC0	1	MCIC byte 0
0E9	NUCMCIC1	1	MCIC byte 1
	NUCMCCP	X'40'	X1XX = channel report pending
0EA	NUCMCIC2	1	MCIC byte 2
0EB	NUCMCIC3	1	MCIC byte 3
0EC	NUCMCIC4	1	MCIC byte 4
0ED	NUCMCIC5	1	MCIC byte 5
0F8	NUCFSA	4	The failing storage address
100	NUCASIT	8	The failing storage asit
120	NUCACRLG	64	The access register save area
160	NUCFPRLG	32	The floating point register save area
180	NUCGPRLG	64	The general purpose register save area
1C0	NUCECRLG	64	The extended control register save area
200	NUCVTAM	4	Reserved for VTAM
204	NUCVMID	8	The virtual machine user ID

Table 5. Contents of the GCS Nucleus Constant Area (NUCON) (continued)

HEX DISP	NAME	LENGTH	DESCRIPTION
20C	NUCLVL	4	The release/service level
20D	NUCRLVL	1	The release level
20E	NUCSLVL	2	The service level
210	NUCIDS	4	The signal ID/task ID
210	NUCSIGID	2	This virtual machine signal ID
212	NUCATID	2	The active task ID
214	NUCATB	4	The address of the active task
218	NUCPOST	4	The branch entry address for the post
21C	NUCCTB	4	The common trace block pointer
220	NUCNPM	4	The network performance monitor
224	NUCSTOR	4	The address of common storage data
228	NUCZIT	4	The start of private storage (dump viewing facility use only)
22C	NUCAGW	4	The AGW RAS use
230	NUCVMPST	4	Reserved for VTAM
234	NUCUSER	4	Reserved for VTAM
238	NUCSAF	4	Reserved for RACF
23C	NUCDUMP	4	Pointer to the dump receiver
240	NUCANCH	4	The pointer to the user anchor table
244	NUCFLAGS	1	FLAGS
	NUCXCMDE	X'80'	XC Virtual Machine
	NUCHWCMP	X'40'	Hardware Compression
28C	NUCFEIBM	12	The component ID-dump viewing facility referenced
298	NUCABW	4	The address of the abend work area (for the dump viewing facility)
29C	NUCRSTS1	4	The system restart save area
2A0	NUCRSTS2	4	The system restart save area
2A4	NUCRSTF	1	The system restart flags
	NUCMSGR	X'02'	The recursion bit (message facility)
	NUCRSTR	X'01'	The recursion bit (restart)
2A5		3	Reserved
2A8	NUCBLRSV	64	The register save area
2E8	NUCCMDLN	160	The command input line
388	NUCCMLST	536	The tokenized PLIST
5A0	NUCUPPER	4	The upper case translate table
5A4	NUCPLFID	4	The flag word used by GCTSCN
5A4	NUCPLSWT	1	The 1-byte switch used in GCTSCN
5A8	NUCCWR	4	The console write routine
5AC	NUCACPF	4	The CP command PASSTHRU
5B0	NUCSCANN	4	The scan routine entry point
5B4	NUCSCNT	4	The scan routine entry point
5B8	NUCPLIST	8	The extended PLIST (untokenized)
5B8	NUCPLCMD	4	The address of the command token
5BC	NUCPLBEG	4	The address of the start of argument string
5C0	NUCPLEND	4	The address of the end of argument string
5C4	NUCSIE	4	The pointer to the SIE (NUCON extension)
5C8	NUCIHCSA	8	The interrupt handler common save area
5D0	NUCSAVQ1	4	The header pointer for the interrupt handler save area
5D4	NUCSAVQ2	4	The trailer pointer for the interrupt handler save area
5D8	NUCSRPTR	4	The pointer to the system restart work area
5DC	NUCDEB	4	The DEB entry to the chain address
5E0	NUCCBLKS	4	The pointer to modules known to program management

Table 5. Contents of the GCS Nucleus Constant Area (NUCON) (continued)

HEX DISP	NAME	LENGTH	DESCRIPTION
5E4		4	A restricted field
650	NUCFCBTB	8	The FCB anchor chain
650	NUCFCB1	4	The address of the first FCB
654	NUCFCBNM	2	The number of FCBs in the chain
658	NUCLAF	4	V(GCTLAF) AACTLKP
65C	NUCERS	4	V(GCTERS) AERASE
660	NUCSTTN	4	V(GCTSTT) AESTATE
664	NUCFNS	4	V(GCTFNS) AFINIS
668	NUCFVS	4	V(FVS) AFVS
66C	NUCAUD	4	V(GCTAUD) AUPDISK
670	NUCRDBUF	4	V(GCTRWBRD) GCTRWBRD
674	NUCDEVTB	4	V(DEVTAB) the address of DEVTAB
678	NUCADTS	4	V(ADTSECT) the address of ADTSECT
67C	NUCDIODA	4	V(DIOSECT) the address of DIODA
680	NUCAFTS	4	V(AFTSTART) the address of AFTSTART
688	NUCTODCA	16	The timing information
688	NUCTODTT	8	The total virtual machine time
690	NUCTODDT	8	The time of day when dispatched
6A0	NUCLLNAM	4	The address of the LOADLIB name list
6A4	NUCLLDIR	4	The address of the LOADLIB directory list
6A8	NUCLLSIZ	4	The size of the LOADLIB name and directory storage
6AC	NUCLLNUM	2	The number of globaled LOADLIBs
6B0	NUCXAWRK	4	The XA-mode work area
6B0	NUCMFLAG	1	The nucleus machine flag:
	NUCXA	X'80'	1XXXXXXX = Virtual machine is XA or XC
	NUCDMPON	X'40'	1X1XXXXX = Dump on switch
	NUCDMPDE	X'20'	XX1XXXXX = Dump default switch
	NUCLCKHO	X'10'	XXX1XXXX = Hold common lock for dump
	NUCIPOLL	X'08'	XXXX1XXX = IPOLL function in use
	NUCNOPLL	X'04'	XXXXX1XX = IPOLL buffer in use
	NUCDMPFM	X'02'	XXXXXX1X = Dump format switch
	NUCSGRP	X'01'	XXXXXXX1 = Single user group
6B1	NUCMASKE	1	The system enable byte
6B2	NUCMASKW	1	The STNSM/STOSM work byte
6B4	NUCLINE	4	The start of high storage
6B8	NUCAMD80	4	Used by GCTAMODE
6BC	NUCAMD7F	4	Used by GCTAMODE
6C0	NUCMCKSA	64	The machine check work area
700	NUCGLUSA	64	The glue work area
740	NUCBESAV	4	The work area
744	NUCBER14	4	Register 14 from the branch entry
	NUCBEA31	X'80'	The branch entry was in AMODE 31
748	NUCFLGS	4	Flags
	NUCREX31	X'80'	REXXSTOR = 31
750	NUCPFPSW	8	The PSW at page fault interruption

SIE—NUCON Extension

Table 6. Contents of the NUCON Extension (SIE)

HEX DISP	NAME	LEN	DESCRIPTION
000	SIE	328	The NUCON Extension
000		8	Eye catcher (GCTSIE)
008	SIETRQ	4	The timer request queue start
00C	SIEQCB	4	The ENQ control block queue start
010	SIETTBL	4	The address of the task ID table
014	SIETBQ	4	The address of the first task block in the dispatch queue
018	SIEAEQ	4	The address of the asynchronous exit queue
01C	SIESCB	4	The pointer to the STAE control block pool
020	SIELKCOM	4	The address of the common storage lock
024	SIELKTID	2	The task ID waiting for the lock
026	SIELOCKB	1	The byte indicating whether the machine
	SIELKCMB	X'80'	is waiting for the lock
027	SIEPM	1	The program management flag byte
	SIEPMGLB	X'80'	Set on when the global LOADLIB command is issued
	*		Set off when the BLDL searches the directories
	SIEPMOSR	X'40'	Set on when OSRUN is active
	*		Set off by LINK
	SIEPMLDR	X'20'	Set on by GCTLOS for LOADADDR
028	SIEVMCBS	4	The address of the VMCB array
02C	SIEVMCB	4	The address of this machine's VMCB
030	SIESYSNM	4	The pointer to the VSAM SYSNAMES table
034	SIEPOST	4	The branch entry to POST
038	SIEGETM	4	The branch entry point to GETMAIN
03C	SIEFREM	4	The branch entry point to FREEMAIN
040	SIESMAB	4	The pointer to the SMAB
044	SIECAADR	4	The address of the attention interrupt ECB
048	SIECIADR	4	The address of the I/O complete ECB
04C	SIECOADR	4	The address of the console output pending ECB
050	SIECTADR	4	The address of the command tack ECB
054	SIECAECB	4	The attention interrupt ECB
058	SIECIECB	4	The I/O complete ECB
05C	SIECOECB	4	The output pending ECB
060	SIECTECB	4	The command task ECB
064	SIECONFL	1	The console task flags
	SIECRDIO	1xxx xxxx	A READ I/O is in progress
	SIECWRIO	x1xx xxxx	A WRITE I/O is in progress
	SIECATTP	xx1x xxxx	The attention pending bit
	SIECOUTP	xxx1 xxxx	The output pending bit
	SIECLEAR	xxxx 1xxx	Clear screen
065	SIECMDFL	3	Reserved command flags
068	SIEFCMDQ	4	The pointer to the first command input buffer
06C	SIELCMDQ	4	The pointer to the last command input buffer
070	SIEFSWQE	4	The pointer to the first WQE buffer on the queue
074	SIELSWQE	4	The pointer to the last WQE buffer on the queue
078	SIEFSORE	4	The pointer to the first ORE buffer on the queue
07C	SIELSORE	4	The pointer to the last ORE buffer on the queue
080	SIECCWS	16	Console CCWS
080	SIECCW1	8	The first CCW
080	SIECCW1C	1	The CCW command code

Table 6. Contents of the NUCON Extension (SIE) (continued)

HEX DISP	NAME	LEN	DESCRIPTION
081	SIECCW1A	3	The data address
084	SIECCW1F	1	A flag byte
085	SIECCW1N	1	An unused flag byte
086	SIECCW1B	2	The byte count
088	SIECCW2	8	The second CCW
088	SIECCW2C	1	The CCW command code
089	SIECCW2A	3	The data address
08C	SIECCW2F	1	A flag byte
08D	SIECCW2N	1	An unused flag byte
08E	SIECCW2B	2	The byte count
090	SIEIDORE	13	The bit string for ORE IDs
09D	SIELSTID	1	The last ID used for assigning
09E		2	Reserved
0A0	SIETAB	4	The trace anchor block pointer
0A4	SIENUCX	4	The pointer to the nucleus extension control block chain
0A8	SIEBVSAM	4	The beginning of the VSAM shared segment
0AC	SIEEVSAM	4	The end of the VSAM shared segment
0B0	SIEBBAM	4	The beginning of the BAM shared segment
0B4	SIEEBAM	4	The end of the BAM shared segment
0B8	SIEIUCAB	4	The IUCV anchor block
0BC	SIESSPTH	2	The signal services path (path ID)
0BE	RESERVED	2	Reserved
0C0	SIEFREST	4	The start of available common free storage
0C4	SIEZNR	4	The start of available private free storage
0C8	SIEVMSIZ	4	The size of this virtual machine
0CC	SIETQE	4	The address of the TQE pool
0D0	RESERVED	4	Reserved for future use
0D4	SIEIFLAG	1	Initialization flags
	SIEPGFT	X'80'	Page faults initialized
	SIEAUSER	X'02'	ON means the virtual machine is authorized
0D5	SIETIME	8	The system save time
0DD	SIEDATE	8	The system save date
0E5	SIECRIT	1	Critical bits
	SIESMGMT	X'80'	Storage management
	SIESTERM	X'40'	System termination
	SIEINIT	X'20'	Initialization
	SIESVC	X'10'	SVC handler
	SIEFSACC	X'08'	File system
	SIEFSERS	X'04'	File system
	SIEFSFNS	X'02'	File system
	SIEFSWRB	X'01'	File system
0E8	SIEREDRN	4	The highest ready task level
0EC	SIEDSP	1	
	SIEDSTOP	X'80'	The priority change bit
0F0	SIESLICE	8	The time slice in microseconds
0F8	SIESDXBR	4	“V(GCTSDXBR)” branch entry to SCHEDEX
0FC	SIESAV	4	“V(GCTSAR)” save area for branch entry
100	SIEIUS	4	“V(GCTIUSBR)” branch entry to IUCV
104	SIEGENIO	4	“V(GCTGIMSB)” branch entry to GENIO START/R
108	SIESATB	4	Saved active task block address
10C	SIESATID	2	Saved active task block ID
10E	SIETRSP	1	Trace service points

Table 6. Contents of the NUCON Extension (SIE) (continued)

HEX DISP	NAME	LEN	DESCRIPTION
	SIETRBRW	X'80'	Trace branch entries to WAIT
	SIETRBRS	X'40'	Trace branch entries to SCHEDEX
	SIETRBRI	X'20'	Trace branch entries to IUCVCOM
	SIETRBRV	X'10'	Trace branch entries to VALIDATE
	SIETRBRP	X'08'	Trace branch entries to POST
10F	SIETRSAV	1	Save trace points
110	SIEASYID	4	“V(SYID)” pointer to the SYID
114	SIEAEXEC	4	“V(GCTREXBR)” pointer to REXX
118	SIEAEXCO	4	“V(GCTREXV2)” pointer to EXECCOMM
11C	SIEAEXGC	4	“V(GCTREXGC)” pointer to GETCOMM
120	SIEAEXSC	4	“V(GCTREXSC)” pointer to SETCOMM
124	SIEMOD	4	“V(GCTMOD):” pointer to GCTMOD
128	SIENTPRI	4	The address of the first Private Level Name/Token pair
12C	SIEFREHC	4	The address of free-high common storage
130	SIEINTAT	4	The address of active task at time of interrupt
134	SIESAI	4	Save area for branch entry
138	SIEIATID	2	The active task id at interrupt
13C	SIEPFECB	4	The address of active page fault ECBs
140	SIEPFFRE	4	The address of free page fault ECBs
144	SIEPFLST	4	The address of last active page fault ECB

TBK—Task Block

Table 7. Contents of Task Blocks

HEX DISP	NAME	LEN	DESCRIPTION
000	TBK	320	The task block
000	TBKUP	4	The address of the task of higher priority
004	TBKDOWN	4	The address of the task of lower priority
008	TBKFRWD	4	The address of the next task of the same priority
00C	TBKBKWD	4	The address of the prior task of same priority
010	TBKACT	4	The active state block address
014	TBKLOAD	4	The load list
018	TBKPSW	8	The PSW loaded by the dispatcher
018	TBKIOMSK	1	The channel and external interrupt masks
019	TBKPKEY	1	The key
	TBKPMXA	X'08'	The XA mode mask
01C	TBKPSWA	4	The second half of the PSW
	TBKPAM31	X'80'	User in AMODE 31
01D	TBKINSTR	3	The instruction address
020	TBKPSW2	4	The last half of PSW for abnormal termination
024	TBKATRSA	4	The address of attach’s register save
028	TBKREGS	64	Registers loaded by dispatcher
068	TBKFLOAT	32	The floating point registers
088	TBKMOM	4	The mother task address
08C	TBKSIB	4	The next task address following the mother task address
090	TBKCHILD	4	The address of the first subtask
094	TBKECB	4	The address of attach ECB posted when subtask completes

Table 7. Contents of Task Blocks (continued)

HEX DISP	NAME	LEN	DESCRIPTION
098	TBKETXR	4	The address of the asynchronous exit block to schedule when task ends
09C	TBKSTAE	4	The address of the ESTAE control block
0A0	TBKDEB	4	The address of the DEB table
0A4	TBKIDENT	4	Machine and task IDs
0A4	TBKMID	2	The machine ID
0A6	TBKTID	2	The task ID 0=Task run in behalf of a user exit called from an interrupt handler 1=Console task 2=Command task
0A8	TBKSTOR	4	The address of the task storage anchor block (TSAB)
0AC	TBKIUCV	4	The address of the IUCV EIB chain
0B0	TBKREXWB	4	The address of the REXX work block
0B4	TBKSFSTL	4	The address of the first line in the program stack
0B8	TBKSLSTL	4	The address of the last line in the program stack
0BC	TBKSNLST	4	The number of lines in the program stack
0C0	TBKSNBST	4	The number of program stacks
0C4	TBKCOMP	4	A task completion code (ABEND)
0C5	TBKCOMP1	3	A completion code value
0C8	TBKRCODE	2	The abend reason code
0CA	TBKKEY	1	The task storage key
0CB	TBKPRIOR	1	The task dispatching priority
0CC	TBKNDSP	1	The task nondispatchability flags
	TBKNODIS	X'80'	The task is non-dispatchable
0CD	TBKFLAG1	1	A flag byte
	TBKPROB	X'80'	The problem state task
	TBKAPPL	X'40'	This is an independent application
	TBKTERM	X'20'	The task has terminated
	TBKNAEB	X'10'	Schedule no AEBs on this task
	TBKESTAE	X'08'	The ESTAE exit routine is active on task
	TBKDUMP2	X'04'	Turned on for the second dump
	TBKDUMP	X'02'	The dump is requested by abnormal termination
	TBKOSACT	X'01'	OSRUN is active on this task
0CE	TBKFLAG2	1	A flag byte
	TBKABEND	X'80'	The abend was entered
	TBKDOS	X'40'	DOS SVC is in effect
	TBKCCVAL	X'20'	The TBKCOMP contains a valid COMP code
	TBKSER	X'10'	GCTSER entered
	TBKFIRST	X'08'	The first task on priority level
	TBKPATHS	X'04'	The IUCV paths defined by the task
	TBKINTER	X'02'	The interrupt task block
	TBKPGFLT	X'01'	The task waiting on page fault
0CF	TBKFLAG3	1	A flag byte
	TBKPGLCK	X'80'	PGLOCK issued for this task
0D0	TBKSUBTA	4	The subtask abend resource manager
0D4	TBKREGSV	4	The address of the abend register save area
0D8	TBKTIME	8	The time task was dispatched
0E0	TBKICODE	2	The interrupt code
0E2	TBKILC1	1	The instruction length
0E3	TBKRXMSK	1	The GCTREX PSW int mask
0E8	TBKRBAD	4	The address of the RB

Table 7. Contents of Task Blocks (continued)

HEX DISP	NAME	LEN	DESCRIPTION
0EC	TBKTIOTA	4	The address of the TIOT
0F0	TBKEPIE	4	The address of the EPIE chain
0F4	TBKWRKEI	4	The address of the EXECIO work area
0F8	TBKWRKCL	4	The address of GCTEIOAB work area
0FC	TBKNTPTR	4	The address of the first Task Level Name/Token pair
100	TBKACRS	64	Access Registers loaded by the dispatcher
140	TBKEND	0	The end of the task block
140	TBKLEN	0	The length of the task block

STBLK—State Block

Table 8. Contents of State Blocks

HEX DISP	NAME	LEN	DESCRIPTION
000	STBLK	240	The state block
000	STBNAME	8	The program name
008	STBPSW	8	The PSW saved for block in STBNEXT
008	STBIOMSK	1	The channel and external interrupt masks
009	STBKCMWS	1	The key, mode, masks, and state
009	STBKEY		The key - bits 0-3
009	STBCMW		The mode, machine check, and wait masks - bits 4-6
	STBEC	X'08'	0=BC mode, 1=XA mode
	STBEM	X'04'	Machine check
	STBEW	X'02'	Wait mask
	STBSTATE	X'01'	0=supervisor, 1=problem state
00A	STBICP	1	XA-Mode ILC, CC, program mask
00C	STBINSTR	4	The instruction address
	STBPSW31	X'80'	AMODE 31 bit
010	STBNEXT	4	The address of the next state block on state stack
014	STBPREV	4	The address of the previous state block - 0 for the first
018	STBTB	4	The address of the task block for this stack
01C	STBNUCBL	4	The address of the NUCCBLK for this module
020	STBENTRY	4	The entry point of the program or SVC
	STBENA31	X'80'	AMODE 31 bit
024	STBFLAG1	1	A flag byte
	STBLINK	X'80'	The link block
	STBSVC	X'40'	The SVC block
	STBAEB	X'20'	The asynchronous exit block (AEB)
	STBSYNCH	X'10'	Synch restore=yes specified
025	STBFLAG2	1	A flag byte
	STBFREE	X'80'	FREEMAIN AEB when the exit ends
	STBGMBR	X'40'	Branch entry (1) for AEB or (0) for SVC entry
	STBAEBSD	X'20'	AEB is for a scheduled exit
	STBAEGIO	X'10'	AEB is for a general I/O
	STBAETIM	X'08'	AEB is for the timer
	STBINTER	X'04'	The interrupt state block
026	STBWAIT	1	The wait count
027	STBMASK	1	The mask at entry to lock
028	STBSP	1	The subpool of GETMAIN for this block
029	STBLDLOS	1	GCTBLDL-GCTLOS communication byte
	STBIOERR	X'80'	ON-BLDL had an I/O error

Table 8. Contents of State Blocks (continued)

HEX DISP	NAME	LEN	DESCRIPTION
02A	STBIORC	1	The I/O error return code
02B	STBLIBCT	1	The LOADLIB number (1 based)
02C	STBICODE	2	The interrupt code
02E	STBILC1	1	The instruction length
02F	STBAMRM	1	The AMODE/RMODE at the time of SVC
	STBCAM31	X'80'	The caller was in AMODE 31
	STBCRM31	X'40'	The caller was in RMODE 31
030	STBEGPRS	64	The caller's register save area (all registers)
070	STBOVER	64	
SECTION FOR ASYNCHRONOUS EXIT AND LINK BLOCKS:			
070	STBWORK	64	The work area
070	STBAETB	4	The task block address used for AE
074	STBAER0	4	The R0 contents when AE gets control
078	STBAER1	4	The R1 contents when AE gets control
07C	STBAER13	4	The R13 contents when AE gets control
080	STBAEPSW	8	The PSW when AE gets control
080	STBAEIOM	1	The channel and external interrupt masks
081	STBAEKC	1	The key, mode, masks and state
081	STBAEKEY		The key - bits 0-3
081	STBAECMW		Mode, machine check, wait masks - bits 4-6
	STBAEC	X'08'	0-BC mode, 1=XA mode
	STBAEM	X'04'	Machine check
	STBAEW	X'02'	The wait mask
	STBAESTA	X'01'	0=supervisor, 1=problem state
084	STBAEINS	4	The instruction address
	STBAEA31	X'80'	AMODE 31 bit
088	STBAEICO	2	The interrupt code
08A	STBAEILC	1	The instruction length
SECTION FOR SVC BLOCKS:			
070	STBSVCA	64	
070	STBRSVD2	1	Reserved
071	STBFLAG3	1	A flag byte
	STBERRET	X'80'	Error return desired
	STBNOSA	X'40'	No save area wanted
	STBRETRG	X'20'	Return callee's R0, R1 to caller
	STBUSVC	X'10'	User SVC call
	STBVSAM	X'08'	OS VSAM request
	STB203	X'02'	SVC 203
	STBOSSIM	X'01'	OS simulation SVC
072	STBCODE	2	The SVC 203 code value
074	STBNRMRT	4	The address of the normal return
078	STBCALLR	4	The address of the SVC caller
07C	STBERADR	4	The address of error return
080	STBEFPRS	32	The caller float register save (0-6)
080	STBEFPR0	8	The caller float register 0 save area
088	STBEFPR2	8	The caller float register 2 save area
090	STBEFPR4	8	The caller float register 4 save area
098	STBEFPR6	8	The caller float register 6 save area
0A0	STBUSAVE	4	A (user save area)
0A4	STBSASZ	2	The size of the user save area

Table 8. Contents of State Blocks (continued)

HEX DISP	NAME	LEN	DESCRIPTION
0A6	STBSAKEY	1	The key of the user save area
0A6	STBSAKY	1	The actual key of the user save area
0A7	STBRSVD3	1	Reserved
0A8	STBOSRS1	4	The first OSRUN save area pointer
0AC	STBOSRS2	4	The second OSRUN save area pointer
COMMON SECTION:			
0B0	STBEACRS	64	The caller's access register save area (all registers)
0F0	STBEND	0	The end of the state block
0F0	STBENDSV	0	The end of the SVC block
0F0	STBSZSVC	0	The length of the SVC block
0F0	STBSZLA	0	The length of the link or AEB

SMAB—Storage Management

Table 9. Contents of Storage Management

HEX DISP	NAME	LEN	DESCRIPTION
000	GCTSMAB	7208	The storage management anchor blocks
000	SMASALT	16	The list of anchor blocks
000	SMALCAB	4	The address of low storage anchor block
004	SMAHCAB	4	The address of high storage anchor block
008	SMALPAB	4	The address of low private storage anchor block
00C	SMAHPAB	4	The address of high private storage anchor block
010	SMATASK	4	The address of the task block of the abending subtask
014	SMAFLAGS	1	Flags
	SMAIPL	X'40'	The IPL initialization is complete
	SMAGFDCP	X'20'	GTCGFDCP is running
	SMACOMMN	X'10'	Getting COMMON storage
	SMANSTK	X'08'	The save area is not on stack (GCTSVQ)
015	SMAGFDFL	1	Used by GCTGFPCP
	SMARTSH	X'80'	We removed a TSH page from the spare list
	SMARGSB	X'40'	We removed a GSB page from the space list
	SMARMNOR	X'20'	We removed a MNOR page from the spare list
	SMARTSBE	X'10'	We removed a TSABE block from the spare list
	SMAATSH	X'08'	We added a TSH page to the spare list
	SMAAGSB	X'04'	We added a GSB page to the spare list
	SMAAMNOR	X'02'	We added a MNOR page to the spare list
018	SMATSBEL	4	Length of the storage used for TSABE
01C	SMATSHF	4	The address of the first page of full TSH pages
024	SMATSHFF	4	First page of the TSH blocks with one 1 free block
02C	SMAGSBF	4	The address of the first page of full GSB pages
034.	SMAGSBFF	4	First page of GSB blocks with 1 free block
03C	SMAGRAIN	4	The size of the grain of storage
040	SMATSHBL	2	The length of a block of TSHs
042	SMATSHBN	2	The number of blocks of TSHs on a page
044	SMATSHBM	2	The maximum number of TSHs in a block
046	SMAGSBBL	2	The length of a block of GSBs
048	SMAGSBBN	2	The number of blocks of GSBs on a page
04A	SMAGSBBM	2	The maximum number of GSBs in a block
04C	SMAFTSH	4	The address of a free TSH page

Table 9. Contents of Storage Management (continued)

HEX DISP	NAME	LEN	DESCRIPTION
054	SMAFGSB	4	The address of a free GSB page
05C	SMAFTSBE	4	The address of free TSABE
060	SMASCOML	4	The address of the start of low common storage
064	SMALCOML	4	The length of low common storage
068	SMASCOMH	4	The address of the start of high common storage
06C	SMALCOMH	4	The length of high common storage
070	SMASAVEA	4	The address of the current GETMAIN/FREEMAIN save area
074	PRISAVEA	2364	Save area set one
074	PRISAVE	60	The register save area for branch entry
0B0	PRIWORK1	256	The work area for branch entry
1B0	PRIWORK2	256	The work area for branch entry
2B0	PRIWORK3	256	The work area for branch entry
3B0	PRIWORK4	256	The work area for branch entry
4B0	PRIWORK5	256	The work area for branch entry
5B0	PRIWORK6	256	The work area for branch entry
6B0	PRIWORK7	256	The work area for branch entry
7B0	PRIWORK8	256	The work area for branch entry
8B0	PRIWORK9	256	The work area for branch entry
9B0	PR2SAVEA	2364	Save area set two
9B0	PR2SAVE	60	The register save area for the second branch entry
9EC	PR2WORK1	256	The work area for branch entry
AEC	PR2WORK2	256	The work area for branch entry
BEC	PR2WORK3	256	The work area for branch entry
CEC	PR2WORK4	256	The work area for branch entry
DEC	PR2WORK5	256	The work area for branch entry
EEC	PR2WORK6	256	The work area for branch entry
FEC	PR2WORK7	256	The work area for branch entry
10EC	PR2WORK8	256	The work area for branch entry
11EC	PR2WORK9	256	The work area for branch entry
12EC	PR3SAVEA	2364	Save area set three
12EC	PR3SAVE	60	The save area for GCTGFDCP
1328	PR3WORK1	256	The work area for branch entry
1428	PR3WORK2	256	The work area for branch entry
1528	PR3WORK3	256	The work area for branch entry
1628	PR3WORK4	256	The work area for branch entry
1728	PR3WORK5	256	The work area for branch entry
1828	PR3WORK6	256	The work area for branch entry
1928	PR3WORK7	256	The work area for branch entry
1A28	PR3WORK8	256	The work area for branch entry
1B28 to 1C27	PR3WORK9	256	The work area for branch entry

ANCH—Storage Anchor Block

Table 10. Contents of Storage Anchor Blocks

HEX DISP	NAME	LENGTH	DESCRIPTION
000	ANCHBK	552	The storage anchor block
000	ANCHFLAG	1	Flags
	ANCHLCAB	X'80'	The anchor block for low common storage

Table 10. Contents of Storage Anchor Blocks (continued)

HEX DISP	NAME	LENGTH	DESCRIPTION
	ANCHHCAB	X'40'	The anchor block for high common storage
	ANCHLPAB	X'20'	The anchor block for low private storage
	ANCHHPAB	X'10'	The anchor block for high private storage
004	ANCHKEYP	512	Starts an array of 32 records, each 4 words long
	ANCHKEYH	4	The head of the SACB queue for this key
	ANCHKEYT	4	The tail of the SACB queue for this key
	ANCHKEYZ	4	The size of the last request under 4K
	ANCHKEYL	4	The SACB of the last request under 4K
204	ANCHPGMN	4	The address of the 1st page of minor SACBs
208	ANCHPGL	4	The major SACB for the lowest fully free page
20C	ANCHPGH	4	The major SACB for the highest fully free page
210	ANCHMAJL	4	The major SACB for the lowest free page of storage
214	ANCHMAJH	4	The major SACB for the highest free page of storage
218	ANCHS200	4	TSABE for storage gotten in subpool 200
21C	ANCHTABL	4	The list of contiguous blocks of free storage
220	ANCHFMNR	4	The free minor's page
224		4	The dummy backward pointer

EXTWA—External Interrupt Handler Work Area

Table 11. Contents of the External Interrupt Handler Work Area (EXTWA)

HEX DISP	NAME	LENGTH	DESCRIPTION
000	EXTWA	328	The external interrupt handler work area
000	EXTPSW	8	The external old PSW
008	EXTSAVE	80	A save area
058	EXTAREA	72	A save area
0A0	EXTREGS	64	Registers at the time of the interrupt
0E0	EXTFPR	32	Floating point registers
100	EXTACRS	64	Access registers at the time of the interrupt
140	EXTICODE	2	The interrupt code
142	EXTILC1	1	The instruction length
143		5	Reserved

SVCWA—SVC Interrupt Handler Work Area

Table 12. Contents of the SVC Interrupt Handler Work Area (SVCWA)

HEX DISP	NAME	LENGTH	DESCRIPTION
000	SVCWA	536	The SVC interrupt handler work area
000	SVCSAVE	64	Registers at the time of the interrupt
040	SVCFREGS	32	Floating point registers
040	SVCFREG0	8	Floating point register 0
048	SVCFREG2	8	Floating point register 2
050	SVCFREG4	8	Floating point register 4
058	SVCFREG6	8	Floating point register 6
060	SVCASAVE	64	Access registers at the time of the interrupt
0A0	SVCSTB	240	The default state block
190	SVCUSA	96	The default user save area
1F0	SVCSTPTR	4	A pointer to the state block in use

Table 12. Contents of the SVC Interrupt Handler Work Area (SVCWA) (continued)

HEX DISP	NAME	LENGTH	DESCRIPTION
1F4	SVCNUM	1	A copy of the SVC number
1F5	SVCILC	1	A copy of the ILC byte
1F6	RESERVED	2	Reserved
1F8	SVCNQRY	24	The PLIST for the NUCEXT QUERY
1F8	SVCNFUNC	8	=CL8'NUCEXT' identifies the NUCEXT function
200	SVCNNAME	8	=CL8' ' nucleus extension name
208	SVCNPTR	4	Receives the pointer to NUCXBLK
20C	SVCNIND	4	=XL4'FFFFFFFF' identifies the NUCEXT QUERY function
210	SVCC14	8	A place for the control reg 14
210	SVCC14B1	1	The first byte of control reg 14
	C14MCKON	X'10'	Enable for CRWs

PGMWA—Program Interrupt Work Area

Table 13. Contents of the Program Interrupt Work Area (PGMWA)

HEX DISP	NAME	LENGTH	DESCRIPTION
000	PGMWA	272	The program check interrupt work area
000	PGMOPSW	8	The program old PSW
008	PGMREGS	64	Registers at time of the interrupt
048	PGMACRS	64	Access registers at time of the interrupt
088	PGMICODE	2	The interrupt code
08A	PGMILC1	1	The instruction length
08B	PGMILCTR	1	ILC for trace
	PGMILCB1	X'02'	ILC bit 1
	PGMILCB2	X'01'	ILC bit 2
090	PGMSAVE	64	The register save area
0D0	PGMPFSAV	64	The page fault reg save area

VMCB—Virtual Machine Control Block

Table 14. Contents of the Virtual Machine Control Block (VMCB)

HEX DISP	NAME	LENGTH	DESCRIPTION
000	VMCB	32	The virtual machine control block
000	VMCUSER	8	The virtual machine user ID
008	VMCINSIG	4	The initialization signal ID
008	*	2	Reserved
00A	VMCSIGID	2	The virtual machine signal ID
00C	VMCLCKH	4	The lock holding pointer
010	VMCLCKW	4	The lock waiting pointer
014	VMCSCHDX	4	The pointer to the chain of AEB blocks to be scheduled for this virtual machine
018	VMCFLAGS	1	Flags
	VMCWAIT	X'80'	The virtual machine in wait

Appendix C. Trace Table Codes

Trace code table entries come in two flavors: 32-byte and 64-byte. The 32-byte entry format is shown in Figure 15. The 64-byte entry format is shown in Figure 16 on page 214.

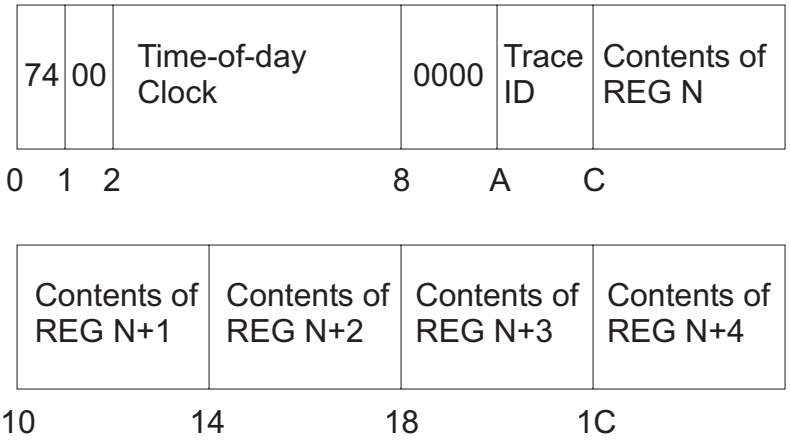


Figure 15. Format of a 32-byte CP Trace Table Entry

Hex Displacement	Contents
X'00'	X'74', which indicates a CP trace table entry.
X'01'	Unused (zeros).
X'02'	The contents of the time-of-day clock at the time the event being traced occurred.
X'08'	Unused.
X'0A'	The trace ID or trace entry code, which defines the event.
X'0C'	The contents of register *n*.
X'10'	The contents of register *n+1*.
X'14'	The contents of register *n+2*.
X'18'	The contents of register *n+3*.
X'1C'	The contents of register *n+4*.

Figure 16 on page 214 illustrates the format of a 64-byte CP trace table entry.

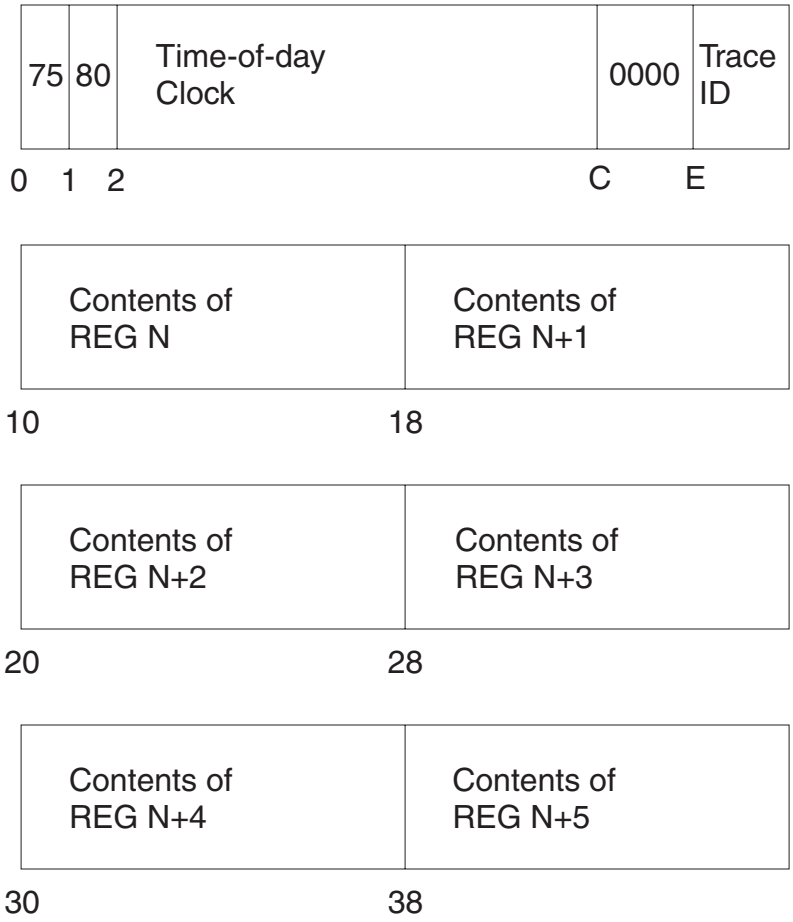


Figure 16. Format of a 64-byte CP Trace Table Entry

Hex Displacement	Contents
X'00'	X'75', which indicates a CP trace table entry.
X'01'	X'80', which indicates a 64-byte CP trace table entry.
X'02'	The contents of the time-of-day clock at the time the event being traced occurred.
X'0C'	Unused.
X'0E'	The trace ID or trace entry code, which defines the event.
X'10'	The contents of register *n*.
X'18'	The contents of register *n+1*.
X'20'	The contents of register *n+2*.
X'28'	The contents of register *n+3*.
X'30'	The contents of register *n+4*.
X'38'	The contents of register *n+5*.

The format of a trace table entry and TRACE ID codes are described by the TTABK. The format of a trace table page and its forward/backward pointers (last two words) are described by the TTPBK.

The following summarizes the event-specific information that CP records in its trace table entries from bytes X'0A' to X'1F' (for 32-byte entries), or from bytes X'0E' to X'3F' (for 64-byte entries).

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
External Interrupt	HCPEXT	0100	00 00 00 00	C'EXT'	Interrupt (PFXEXTCP)	External Old PSW (PFXEXTOP)	
SVC Interrupt	HCPSVC	0200	Abend if Abend C'MM', X'NN'	C'SVC'	SVC Interrupt (PFXSVCIL)	SVC Old PSW (PFXSVCOP)	
Program Interrupt	HCPPRG	0300	VMDBK Address or 00 00 00 00	ILC, Interrupt	Translation Exception Address or 00 00 00 00	Program Old PSW	
Machine Check Interrupt	HCPMCH	0400	Failing Storage Address (FSA)	Machine Check Interruption Code (MCIC)		Machine Check Old PSW	
I/O Interrupt	HCPIFI	0500	RDEVDEV|RDEVSUB	Real Device Block Address	PFXRNUSR	I/O Old PSW	
		0501	See Format 2 table				
I/O Interrupt type 3 subchannel	HCPIFH	0530	E3 F3|VDEVSUB	Real Device Block Address	PFXRNUSR	I/O Old PSW	
Obtain Free Storage (Free)	HCPFRE HCPFRF	0600	Block's ID'<xxx'	Doublewords Requested (PFXFRERO)	Address of Assigned Block (GPR2)	VMDBK Address (GPR11)	Caller's Return Address (PFXFRERE)
Obtain Pageable Free Storage	HCPVFM	0610	Block's ID'<xxx'	Double Words Requested	Assigned Block Address (virtual)	Requestor's VMDBK Address (GPR11)	Caller's Return Address (REG14)
Obtain SCSI Pool Storage	HCPFRX	0620	Block's Alignment	Bytes Requested	Assigned Block Address	Requestor's VMDBK Address	Caller's Return Address
Return Free Storage (FRET)	HCPFRE HCPFRF	0700	Block's ID'<xxx'	Doublewords Returned (GRP0)	Returned Block Address (GPR1)	Caller's VMDBK Address (GPR11)	Caller's Return Address (GPR14)
Return Pageable Free Storage	HCPVFM	0710	Block's ID'<xxx'	Doublewords Returned (GRP0)	Returned Virtual Block Address (GPR1)	Caller VMDBK Address (GPR2)	Caller's Return Address (GPR14)
Return SCSI Pool Storage	HCPFRX	0720	00 00 00 00	Bytes Returned	Returned Block Address	Caller VMDBK Address	Caller's Return Address
Run User	HCPRUN	0A00	00 00 00 00	00|C'RUN'	VMDBK Address	Guest PSW (VMDPSW)	
Virtual XA I/O Interrupt	HCPVIS	0C00	User I/O Old PSW (UZPIOOP)		VDEVDEV|RDEVDEV	Subchannel ID (SID)	Interrupt Parameter
		0C01	See Format 2 table				
Virtual Adapter Interruption	HCPVIS	0C02	I/O Old PSW Bytes 0–3	I/O Old PSW Bytes 4–7	VMDBK	Interruption ID Word	

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Virtual Modify Subchannel	HCPVOL	0C32	VDEVDEV|RDEVDEV	Path Management Control Words (PMCW) 0-3			
Virtual Start Subchannel	HCPVOS	0C33	VDEVDEV|RDEVDEV	Operand (VMDICAD1)	Operation Request Block (ORB)		
Virtual Test Subchannel	HCPVOS	0C35	VDEVDEV|RDEVDEV	Subchannel Status Word (SCSW) (IORSCSW)			Extended Status Word (ESW)
Virtual Test Pending Interrupt	HCPVOS	0C36	00 00|VMDINST	Operand (VMDICAD1)	VDEVDEV|RDEVDEV	Subchannel ID (SID)	Interrupt Parameter
Virtual CSW Stored	HCPCSW	0D00	00 00|VMDINST	VDEVDEV|RDEVDEV	Limited Channel Logout	Channel Status Word (CSW)	
Virtual Start I/O	HCPVOD HCPVOH	0D90	00 00|VMDINST	VDEVDEV|RDEVDEV	Channel Address Word (CAW)	First CCW in Channel Program	
Virtual Start I/O Fast	HCPVOD	0D91	00 00|VMDINST	VDEVDEV|RDEVDEV	Channel Address Word (CAW)	First CCW in Channel Program	
Virtual XA I/O Interrupt (type 3 subchannel)	HCPVIS	0E00	User I/O Old PSW (UZPIOOP)		00 00|E3 F3	Subchannel ID (SID)	Interrupt Parameter
Virtual Modify Subchannel (type 3)	HCPVOL	0E32	00 00|E3 F3	Path Management Control Words (PMCW) 0-3			
Virtual Start Subchannel (type 3)	HCPVOS	0E33	00 00|E3 F3	Operand (VMDICAD1)	Operation Request Block (ORB)		
Virtual Test Subchannel (type 3)	HCPVOS	0E35	00 00|E3 F3	Subchannel Status Word (SCSW) (IORSCSW)			Extended Status Word (ESW)
Virtual Test Pending Interrupt (type 3 subchannel)	HCPVOS	0E36	00 00|VMDINST	Operand (VMDICAD1)	00 00|E3 F3	Subchannel ID (SID)	Interrupt Parameter
Clear Subchannel, CC=0	HCPIOS	1000	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Clear Subchannel, CC=3	HCPIOS	1003	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Halt Subchannel, CC=0	HCPIOS	1010	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Halt Subchannel, CC=1	HCPIOS	1011	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Halt Subchannel, CC=3	HCPIOS	1013	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Modify Subchannel, CC=0	HCPIOS	1020	RDEVDEV|RDEVSUB	Path Management Control Words (PMCW) 1-3			PMCW Word 6
Modify Subchannel, CC=1	HCPIOS	1021	RDEVDEV|RDEVSUB	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Modify Subchannel, CC=3	HCPIOS HCPVOL	1023	RDEVDEV|RDEVSUB	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Start Subchannel, CC=0	HCPIOS	1030	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	Operation Request Block (ORB)		
Start Subchannel, CC=1	HCPIOS	1031	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Subchannel, CC=3	HCPIOS	1033	RDEVDEV|RDEVSUB	IORBK Address	Operation Request Block (ORB)		
Start Subchannel, CC=0 Sense	HCPIFI	1038	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	Operation Request Block (ORB)		
Start Subchannel, CC=1 Sense	HCPIFI	1039	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Subchannel, CC=3 Sense	HCPIFI	103B	RDEVDEV| RDEVSUB	IORBK Address	Operation Request Block (ORB)		
Test Subchannel, CC=0	HCPIFI	1050	RDEVDEV|RDEVSUB	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Test Subchannel, CC=1	HCPIFI	1051	RDEVDEV|RDEVSUB	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Test Subchannel, CC=3	HCPIFI	1053	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Real Resume Subchannel, CC=0	HCPPAH HCPPAU HCPVOD HCPVOS HCPVIR	1080	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	Previous Suspended PIOBK Host Absolute Address[1]	RDCBK Address[1]	CPVOL Address[1]

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Real Resume Subchannel, CC=1	HCPPAH HCPPAU HCPVOD HCPVOS HCPVIR	1081	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	Previous Suspended PIOBK Host Absolute Address[1]	RDCBK Address[1]	CPVOL Address[1]
Real Resume Subchannel, CC=2	HCPPAH HCPPAU HCPVOD HCPVOS HCPVIR	1082	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	Previous Suspended PIOBK Host Absolute Address[1]	RDCBK Address[1]	CPVOL Address[1]
Real Resume Subchannel, CC=3	HCPPAH HCPPAU HCPVOD HCPVOS HCPVIR	1083	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	Previous Suspended PIOBK Host Absolute Address[1]	RDCBK Address[1]	CPVOL Address[1]
Channel Information Request/Response	HCPCIO	1090	Request Data		Response Data		
Type 1 Subchannel Test Subchannel, CC=0	HCPIFH	1091	RDEVSID	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Type 1 Subchannel Test Subchannel, CC=1	HCPIFH	1092	RDEVSID	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Type 1 Subchannel Test Subchannel, CC=3	HCPIFH	1093	RDEVSID	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Type 1 Subchannel Modify Subchannel, CC=0	HCPIID	1095	RDEVSID	Path Management Control Words (PMCW) 1-3			PMCW Word 6
Type 1 Subchannel Modify Subchannel, CC=1	HCPIID	1096	RDEVSID	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Type 1 Subchannel Modify Subchannel, CC=2	HCPIID	1097	RDEVSID	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Type 1 Subchannel Modify Subchannel, CC=3	HCPIID	1098	RDEVSID	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Type 1 Subchannel Interrupt	HCPIFH	1099	RDEVSID	Real Device Block Address	PFXRNUSR	I/O Old PSW	
Cancel I/O Request	HCPIOX	10A0	RDEVDEV|RDEVSUB	IORBK Address	Return Code	00 00 00 00	00 00 00 00
I/O Sense Data Received	HCPIFI	10F0	RDEVDEV|IORSCNT	Sense Data Bytes 0 - 15			
Concurrent Sense Data Received	HCPIFI	10F1	RDEVDEV ERW Byte 0 IORERWCT	Concurrent Sense Data Bytes 0 - 15			
Clear Subchannel, CC=0 (type 3)	HCPIOS	1300	E3 F3|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Clear Subchannel, CC=3 (type 3)	HCPIOS	1303	E3 F3|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Halt Subchannel, CC=0 (type 3)	HCPIOS	1310	E3 F3|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Halt Subchannel, CC=1 (type 3)	HCPIOS	1311	E3 F3|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Halt Subchannel, CC=3 (type 3)	HCPIOS	1313	E3 F3|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Modify Subchannel, CC=0 (type 3)	HCPIOS	1320	E3 F3|RDEVSUB	Path Management Control Words (PMCW) 1-3			PMCW Word 6
Modify Subchannel, CC=1 (type 3)	HCPIOS	1321	E3 F3|RDEVSUB	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Modify Subchannel, CC=3 (type 3)	HCPIOS HCPVOL	1323	E3 F3|RDEVSUB	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Start Subchannel, CC=0 (type 3)	HCPIOS	1330	E3 F3|RDEVSUB	Active IORBK Address (RDEVAIOR)	Operation Request Block (ORB)		
Start Subchannel, CC=1 (type 3)	HCPIOS	1331	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Subchannel, CC=3 (type 3)	HCPIOS	1333	E3 F3|RDEVSUB	IORBK Address	Operation Request Block (ORB)		
Test Subchannel, (type 3)	HCPIFH	1350	E3 F3|RDEVSUB	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Test Subchannel, CC=1 (type 3)	HCPIFH	1351	E3 F3|RDEVSUB	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Test Subchannel, CC=3 (type 3)	HCPIFH	1353	E3 F3|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Type 3 Subchannel Cancel I/O Request	HCPIOX	13A0	E3 F3|RDEVSUB	|ORBK Address	Return Code	00 00 00 00	00 00 00 00
APPC/VM SENDXXXX	HCPIUA	1404	Address of the IUCVB	Path ID|Rtn Code| Flags1	State|Flags2|What RC| SENDOP	Address of MSGBK	Address of Next Instruction
APPC/VM RECEIVE	HCPIUA	1405	Address of the IUCVB	Path ID|Rtn Code| Flags1	State|Flags2|What RC| SENDOP	Address of MSGBK	Address of Next Instruction
APPC/VM CONNECT	HCPIUA	140B	Address of the IUCVB	Path ID|Rtn Code| Flags1	State|Flags2|What RC| 00	00 00 00 00	Address of Next Instruction
APPC/VM SEVER	HCPIUA	140F	Address of the IUCVB	Path ID|Rtn Code| Flags1	State|Flags2|What RC| 00	00 00 00 00	Address of Next Instruction
APPC/VM QRYSTATE	HCPIUA	1412	Address of the IUCVB	Path ID|Rtn Code| Flags	State|FLAGS2|00| 00	SIP Code| SIP Flag| SPC Mod| SYNC Level	Address of Next Instruction
APPC/VM SETMODFY	HCPIUA	1413	Address of the IUCVB	Path ID|Rtn Code| Flags	State|00 00 00	SENDOP2| 00 00 00	Address of Next Instruction
APPC/VM SETSTATE	HCPIUA	1414	Address of the IUCVB	Path ID|Rtn Code| Flags	State|00 00 00	SENDOP2|00| 00 00	Address of Next Instruction
APPC/VM CONNECT Resume Suspended Connect	HCPIUR	142B	Address of the IUCVB	Path ID|Rtn Code|00	Flags1|Flags2| What RC|00	IPR Code|00 00	Address of Next Instruction
APPC/VM Interrupt	HCPIUM	1430	Address of the IUCVB	Path ID|Int Type| Flag	CCTBK Address	IPARML Address	VMDBK Address
APPC|VM CONNECT Resume Unnecessary	HCPIUR	143B	00 00 00 00	00 00 00 00	00 00 00 00	User ID	
IUCV Query	HCPIUA	1500	Address of the IUCVB	00 00 00 00	CP-SYSCD|00 00 00	Parmsize|Max. No. of Connections	Address of Next Instruction
IUCV Test Message	HCPIUA	1501	Address of the IUCVB	CC|00 00 00	00 00 00 00	00 00 00 00	Address of Next Instruction
IUCV Retrieve Buffer	HCPIUA	1502	Address of the IUCVB	00 00 00 00	CP-SYSCD|00 00 00	Address of Buffer	Address of Next Instruction
IUCV Describe	HCPIUA	1503	Address of the IUCVB	Path ID|Rtn Code| Flags	00 00 00 00	Address of MSGBK	Address of Next Instruction
IUCV Send	HCPIUA	1504	Address of the IUCVB	Path ID|Rtn Code| Flags	CP-SYSCD|00 00 00	Address of MSGBK	Address of Next Instruction
IUCV Receive	HCPIUA	1505	Address of the IUCVB	Path ID|Rtn Code| Flags	CP-SYSCD|00 00 00	Address of MSGBK	Address of Next Instruction
IUCV Reply	HCPIUA	1506	Address of the IUCVB	Path ID|Rtn Code| Flags	CP-SYSCD|00 00 00	Address of MSGBK	Address of Next Instruction

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
IUCV Test Completion	HCPIUA	1507	Address of the IUCVB	Path ID|Rtn Code| Flags	00 00 00 00	Address of MSGBK	Address of Next Instruction
IUCV Reject	HCPIUA	1508	Address of the IUCVB	Path ID|Rtn Code| Flags	CP-SYSCD|00 00 00	Address of MSGBK	Address of Next Instruction
IUCV Purge	HCPIUA	1509	Address of the IUCVB	Path ID|Rtn Code| Flags	CP-SYSCD|00 00 00	Address of MSGBK	Address of Next Instruction
IUCV Accept	HCPIUA	150A	Address of the IUCVB	Path ID|Rtn Code| Flags1	CP-SYSCD²|Flags2² 00 00	00 00 00 00	Address of Next Instruction
IUCV Connect	HCPIUA	150B	Address of the IUCVB	Path ID|Rtn Code| Flags	CP-SYSCD|00 00 00	00 00 00 00	Address of Next Instruction
IUCV Declare Buffer	HCPIUA	150C	Address of the IUCVB	00 00|Rtn Code|00	CP-SYSCD|00 00 00	Address of the Buffer	Address of Next Instruction
IUCV QUIESCE	HCPIUA	150D	Address of the IUCVB	Path ID|Rtn Code| Flags	CP-SYSCD|00 00 00	00 00 00 00	Address of Next Instruction
IUCV Resume	HCPIUA	150E	Address of the IUCVB	Path ID|Rtn Code| Flags	CP-SYSCD|00 00 00	00 00 00 00	Address of Next Instruction
IUCV Sever	HCPIUA	150F	Address of the IUCVB	Path ID|Rtn Code| Flags	CP-SYSCD|00 00 00	00 00 00 00	Address of Next Instruction
IUCV Set Mask	HCPIUA	1510	Address of the IUCVB	Mask|00 00 00	00 00 00 00	00 00 00 00	Address of Next Instruction
IUCV Set Control Mask	HCPIUA	1511	Address of the IUCVB	Mask|00 00 00	00 00 00 00	00 00 00 00	Address of Next Instruction
IUCV IPOLL	HCPIUA	1515	Address of the IUCVB	00 00 00 00	Buffer Length| Data Length	Address of the data buffer	Address of Next Instruction
IUCV Interrupt	HCPIUM	1530	Address of the IUCVB	Path ID|Int Type| Flag	CCTBK Address	IPARML Address	VMDBK Address
IUCV System Service IXBLK	HCPIUF	1531	CSS ID|00 00 00	CSS Path|Ipr Code| Flag	IXBLOK Address	MSGBK Address	IRA Address
IUCV No Interrupt	HCPIUM	1550	Address of the IUCVB	NONE	CCTBK Address	IPARML Address	VMDBK Address
CCS Accept	HCPVCT HCPVCW	1600	Accept Data, bytes 8 thru 15		Path ID (CCS)|00 00	Accept Data, bytes 0 thru 7	
CCS PURGE	HCPVCW	1603	00 00 00 00	00 00|IP-RCODE| 00	Path ID (CCS)|00 00	Address of the RDEV	Current VMDBK Address
CCS RECEIVE	HCPVCP	1604	Address of the SNABK	00 00|WEB-MODE| WEB-LAID	Path ID (CCS)| Path ID (VSM)	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
CCS REPLY	HCPVCV	1606	Address of the SNABK	00 00|WEB-MODE| WEB-LAID	Path ID (CCS)| Path ID (VSM)	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK
CCS SEND 1-way	HCPVCV HCPVCX	1608	Address of the SNABK	00 00|WEB-MODE| WEB-LAID	Path ID (CCS)| Path ID (VSM)	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK
CCS SEND 2-way	HCPVCV HCPVCX	1609	Address of the SNABK	00 00|WEB-MODE| WEB-LAID	Path ID (CCS)| Path ID (VSM)	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK
CCS SEVER	HCPVCT HCPVCX HCPVCZ	160A	User Data|Netname, bytes 1 thru 7		Path ID (CCS)| Path ID (VSM)	Luname	
CCS Logic Error in CCS WEBBK	HCPVCP	160B	Address of the SNABK	00 00|WEB-MODE| WEB-LAID	Path ID (CCS)| Path ID (VSM)	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK
CCS REPLY from VSM	HCPVCQ HCPVCR HCPVCS HCPVCW HCPVCY	160C	Address of the SNABK	00 00|WEB-MODE| WEB-LAID	Path ID (CCS)| Path ID (VSM)	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK
CCS CONNECT for VSM	HCPVCT	160E	00 00 00 00	00 00|Timer	Path ID (CCS)| MSGLIMIT	VSM Userid	
CCS SEVER from VSM	HCPVCT	1610	00 00 00 00	00 00|User Data|00	Path ID (CCS)|00 00	VSM Userid	
CCS Message Complete	HCPVCQ	1611	Address of the SNABK	00 00 00|WEB-MODE	Path ID (CCS)| Path ID (VSM)	00 00|IP-AUDIT1| IP-AUDIT2	Address of IUCV IXBLK
CCS CONNECT for LU	HCPVCT	1612	Netname		Path ID (CCS)|00 00	Luname	
CCS Logic Error in VSM WEBBK	HCPVCX	1613	Address of the SNABK	00|00|WEB-MODE| WEB-LAID	Path ID (CCS)| Path ID (VSM)	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
CCS Error in User Environment	HCPVCP	1614	00 00 00 00	00 00|WEB-MODE| WEB-LAID	Path ID (CCS)|00 00	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK
CCS Soft Abend VCX002	HCPVCX	1615	00 00 00 00	00 00 00 00	00 00 00 00	Address of Last Instruction	Current VMBDK Address
CCS ACCEPT Error	HCPVCT HCPVCW	1680	Accept Data, bytes 8 thru 15		Path ID (CCS)| IP-RCODE|00	Accept Data, bytes 0 thru 7	
CCS RECEIVE Error	HCPVCP	1684	Address of the SNABK	IP-RCODE|00| WEB-MODE| WEB-LAID	Path ID (CCS)| Path ID (VSM)	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK
CCS REPLY Error	HCPVCV	1686	Address of the SNABK	IP-RCODE|00| WEB-MODE| WEB-LAID	Path ID (CCS)| Path ID (VSM)	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK
CCS SEND 1-Way Error	HCPVCV HCPVCX	1688	Address of the SNABK	IP-RCODE|00| WEB-MODE| WEB-LAID	Path ID (CCS)| Path ID (VSM)	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK
CCS SEND 2-Way Error	HCPVCV HCPVCX	1689	Address of the SNABK	IP-RCODE|00| WEB-MODE| WEB-LAID	Path ID (CCS)| Path ID (VSM)	WEB-FUN| WEB-CPFLG| WEB-EDIT| WEB-CHAR	Address of the IUCV IXBLK
Queue State Change / Data Transfer	HCPVQO	1700	RDEVDEV/0000	QIOBK	PREV State NEW State Flags 00	QDIO Buffer Number /0000	
Queue State Change / Error	HCPVQO	1701	RDEVDEV/0000	QIOBK	PREV State NEW State Flags 00	QDIO Buffer Number /0000	Error Report Word
SIGA CC 0 (Real)	HCPIOT	1702	VDEVDEV/RDEVDEV	Function Code	QDIO Queue Mask (R2)	QDIO Queue Mask (R3)	
SIGA CC 1 (Real)	HCPIOT	1703	VDEVDEV/RDEVDEV	Function Code	QDIO Queue Mask (R2)	QDIO Queue Mask (R3)	
SIGA CC 3 (Real)	HCPIOT	1704	VDEVDEV/RDEVDEV	Function Code	QDIO Queue Mask (R2)	QDIO Queue Mask (R3)	
SIGA CC 0 (Virtual)	HCPVQO	1705	Subchannel ID (SID)	Function Code	QDIO Queue Mask (R2)	QDIO Queue Mask (R3)	
SIGA CC 1 (Virtual)	HCPVQO	1706	Subchannel ID (SID)	Function Code	QDIO Queue Mask (R2)	QDIO Queue Mask (R3)	

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
SIGA CC 3 (Virtual)	HCPVQO	1707	Subchannel ID (VDEVSID)	Function Code	QDIO Queue Mask (R2)	QDIO Queue Mask (R3)	
SIGA CC 2 (Real)	HCPIOT	1708	VDEVDEV/RDEVDEV	Function Code	QDIO Queue Mask (R2)	QDIO Queue Mask (R3)	
SIGA CC 2 (Virtual)	HCPVQO	1709	Subchannel ID (SID)	Function Code	QDIO Queue Mask (R2)	QDIO Queue Mask (R3)	
Adapter Interruption	HCPVAI	170A	Interruption ID Word	Time Delay Interval	PFXRNUSR	I/O Old PSW Bytes 0–3	I/O Old PSW Bytes 4–7
Poll for Real AIF Events	HCPVAI	170B	VPHDR	First VPBLK on the chain	VMDBK		Time Delay Value
Defer AIF Polling	HCPVAI	170C	VPHDR	First VPBLK on the chain	VMDBK		Time Delay Value
Run User in Virtual SIE Mode	HCPWRU	1A00	00 00 00 00	00|C'WRU'	R/Guest VMDBK Address	V/Guest PSW (VMDPSW)	
Virtual SIE Interception	HCPWRU	1A11	00 00 00 00	VMD-ICODE| VMD-ICFLG| VMDINST	Operand Address (VMDICAD1)	V|Guest PSW (VMDPSW)	
Unit Check	HCPTRE	1C01	RDEVDEV|RDEVSUB or 00 00	IO-RECLVL| IO-RTYGBL|	IOR-FLAG|00 00 00	Failing CCW	
Unit Exception	HCPRDE	1C02	RDEVDEV|RDEVSUB or 00 00	IO-RECLVL| IO-RTYGBL (GPR1)	IOR-FLAG|00 00 00 (GPR2)	Failing CCW (GPR3 and 4)	
I/O Related Machine Check	HCPRFC	1D01	00 00 00 00	00 00 00 00	Channel Report Word (CRW)	Machine Check Input Parameter or 00 00 00 00	00 00 00 00
Channel Check	HCPRFC	1D02	RDEVDEV|RDEVSUB or 00 00	Real Device Block Address or 00 00 00 00	System Log Error Record Address	00 00 00 00	00 00 00 00
Channel Check at Termination	HCPRFC	1D03	00 00 00 00	00 00 00 00	PFXMCHIN or 00 00 00 00	00 00 00 00	Channel Report Word (CRW)
Store CRW (STCRW)	HCPCPR	1D04	00 00 00 00	00 00 00 00	CRW Value	CHPID	Path Initialized=0 Path Not Initialized=4
ADD or STACK COMBK	HCPQCO HCPVCQ HCPVCR HCPVCS	2200	COMBK Address	COM-STAT| COM-DFLAG| COM-BPARM| COM-PARM	Originator VMDBK	Destination VMDBK	RDEVBK Address (GPRB)

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Execute CP Command	HCPCFM	2301	00 00 00 00	00 00|GSD-TYPE| GSD-FLAG	BASE VMDCFCTL| VMDCFCTL|BASE VMDCWAIT| VMDCWAIT	Abbreviated Command	
Guest I/O Untranslation	HCPUNT	2500	00 00 00 00	'UNT'	CCW Address	Guest Translated CCW	
Call-With-Savearea	HCPSVC	2800	PARM REG (GPR2)	Caller Module ID C'XXX'	New SAVBK Address (GPR13)	Caller Real Address (SAVER14)	Callee Virtual Address (SAVER15)
Indirect Call Request	HCPSVC	2810	PARM REG (GPR2)	Caller Module ID, C'xxx'	New SAVBK Address (GPR13)	Caller Real Address (SAVER14)	Callee Real ICRBK Address (SAVER15)
Local Call Request	HCPSVC	2820	PARM REG (GPR2)	Caller Module ID, C'xxx'	New SAVBK Address (GPR13)	Caller Real Address (SAVER14)	Callee Real Address (SAVER15)
C Function Call	(Various)	2890	00 00 00 00	Caller Real Address	Savearea (SZVSB)	Callee Real Address	Function Name Address
C Function Call in Device Driver	(Various)	28A0	Address of Function Name Address	Caller Real Address	Savearea (SZVSB)	Callee Real Address	00 00 00 00
Return-With-Savearea	HCPSVC	2C00	Return (SAVER15)	CC & Return Module Prog ID C'XXX' Mask	Returned SAVBK Address (GPR13)	Caller Real Address Address (SAVER14)	Callee Real Exit (PFXLNK14)
Indirect Call Return	HCPSVR	2C10	Return Code (SAVER15)	CC & Return Module ID (C'xxx')	Returned SAVBK Address (GPR13)	Caller Real Address (SAVE14)	Callee Real Exit (PFXLNK14)
Local Call Return	HCPSVR	2C20	Return Code (SAVER15)	CC & Return Module ID (C'xxx')	Returned SAVBK Address (GPR13)	Caller Real Address (SAVE14)	Callee Real Exit (PFXLNK14)
C Function Return	(Various)	2C90	Function Header Address	Caller Real Address	Savearea (SZVSB)	Callee Real Address	00 00 00 00
		2CA0	See Format 2 table				
C Function Return in Device Driver	(Various)	2CB0	Address of Function Name Address	Caller Real Address	Savearea (SZVSB)	Callee Real Address	R15
Stack IORBK/TRQBK	HCPSTK	3000	C'KI '	00|VMD-STATE| VMD-SLIST|||T[3]	VMDBK Address Address	IORBK|TRQBK Requesting Stack Address	Calling routine
Unstack IORBK/TRQBK	HCPDSB	3010	C'UI '	00|00|00|||T[3]	VMDBK Address	IORBK|TRQBK Address	IORIRA|TRQBIRA (Exit Address)
Add User to Dispatch List	HCPSTK	3200	VMDWSSPR -or- VMDHOTWS	VMD-TYPE| VMD-STATE|00| VMD-QSTAT	VMDBK Address	VMDDPRTY (Initial Value)	

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Drop User from Dispatch List	HCPSTK	3210	VMDWSSPR -or-VMDHOTWS	VMD-TYPE|VMD-STATE|00|VMD-QSTAT	VMDBK Address	VMDDPRTY (Final Value)	
Stack CPEBK	HCPSTK	3300	C'CK '	00|VMD-STATE|VMD-SLIST|CPEX-SCHC	VMDBK Address	CPEBK Address	Calling Routine Requesting Stack Address
Unstack CPEBK	HCPDSB HCPCFM	3310	C'UC '	00 00 00|CPEX-SCHC	VMDBK Address	CPEBK Address	CPEXR15 (Exit Address)
Interception, Not Instruction	HCPRUN	3500	00 00 00 00	00 00 00|VMD-ICODE	00 00 00 00	Guest PSW (VMDPSW)	
Instruction Interception	HCPPRV	3504	Contents of Guest GPR1	00|VMD-ICLFG|VMDINST	Operand Address	Guest PSW (VMDPSW)	
Exit to the Dispatcher	HCPDSP	3600	VMD-STATE|VMD-IDLCTL|VMD-RSTAT|VMD-DWFLG	00|Exiting Module ID C'XXX'	Address of Currently Selected VMDBK	00 00 00 00	Exiting Module Address
Enter enabled wait state	HCPWAI	3650	0	Mask of waiting CPUs (SRMWTCPU)	0	0	0
Stack Work Bits	HCPSTK	3700	C'K W00'	00|VMD-STATE|VMD-SLIST|00	VMDBK Address	Work Bit Being Stacked	Calling Routine Requesting Stack Address
Processor Controller Diagnose Request Started	HCPPCA	3C00	Command Word (PCRCMDWD)	Base VMDBK (PCRVMBAS) DIAG CC (byte 3)	PC Data Block Address A(HCPWRKPC)	PC Request Block Address PCSPCRQ	PC Status Block Address (SYSPCSBK)
Processor Controller Service Call Request Started	HCPPCB	3C10	Command Word (PCRCMDWD)	Base VMDBK (PCRVMBAS) DIAG CC (byte 3)	PC Data Block Address A(HCPWRKPC)	PC Request Block Address (PCSPCRQ)	PC Status Block Address (SYSPCSBK)
Processor Controller Diagnose Request Returned	HCPPCA	3C55	00 00|PCD-RESPD|PCD-RESPS	Base VMDBK Address (PCRVMBAS)	PC Data Block Address (SAVEWRK1)	PC Request Block Address (PCSREQBK)	PC Status Block Address (SYSPCSBK)
Processor Controller Service Request Returned	HCPPCB	3C65	00 00|PCD-RESPD|PCD-RESPS	Base VMDBK Address (PCRVMBAS)	PC Data Block Address (PCRRDBKA)	PC Request Block Address (PCSREQBK)	PC Status Block Address (SYSPCSBK)
Unsolicited Processor Controller Interrupt Received	HCPPCR	3CFF	00 00 00 00 (GPR4)	00 00 00 00 (GPR5)	PC Data Block Address (SAVER1) (GPR6)	00 00 00 00 (GPR7)	PC Status Block Address (SYSPCSBK) (GPR8)
PTR Translation Results	HCPPTR	4000	Virtual Address (GPR1)	Real Address (GPR2)	VMDBK Address Translated for (GPR10)	Dispatched VMDBK Address (GPR11)	Address of Caller (SAVER14)

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
		4003	See Format 2 table				
		4004	See Format 2 table				
Virtual Machine Storage Locking	HCPVMS	4010	00XA or 0370	Caller's Address	VMDBK Address	Virtual Address to Lock	Real Frame's Address
Virtual Machine Storage Unlocking	HCPVMS	4011	00XA or 0370	Caller's Address	VMDBK Address	Virtual Address to Unlock	Real Frame's Address
		4020	See Format 2 table				
Minidisk Cache Page Fault	HCPFTP HCPFMP	4100	Fault Address In Cache	Task Block Address TSKBK	AR Number|Flags| Operand1|Operand2	VDEV Address	TCHBK Address
Minidisk Cache Miss Read Complete, CC=0	HCPFTH	4101	Track Format Data TCHTKFMT	Hash Key TCHKEY	Hash Table Entry TCHBK Address	STKFLAG| PROCFLAG|0| PROCFLG2	SAVBK Address (ESA/390) SVGBK Address (SVGBK)
Minidisk Cache Miss Read Failed, CC=3	HCPFTH	4102	VDEV Address	Hash Key TCHKEY	Hash Table Entry TCHBK Address	STKFLAG| PROCFLAG| REASNCOD| PROCFLG2	SAVBK Address (ESA/390) SVGBK Address (z/Arch)
Minidisk Cache Miss I/O Needed	HCPFTH	4103	VDEV Address	Hash Key TCHKEY	Hash Table Entry TCHBK Address	STKFLAG| PROCFLAG|0| PROCFLG2	SAVBK Address (ESA/390) SVGBK Address (z/Arch)
Minidisk Cache Miss I/O Complete	HCPFTI HCPFMI	4104	Track Format Data TCHTKFMT	Hash Key TCHKEY	Hash Table Entry TCHBK Address	Cache Address Space ID TCHADRLX	SAVBK Address (ESA/390) SVGBK Address (z/Arch)
Minidisk Cache, Deferral	HCPFTH	4105	Resume Address	Hash Key TCHKEY	Hash Table Entry TCHBK Address	STKFLAG| PROCFLAG|0| PROCFLG2	CPEBK Address CNOCPEBK (SAVBK)
MDC Main Steal Complete	HCPFTV HCPFMV	4106	Target Size for Main Cache	Actual Main Cache Size (TCMMAIN)	Number Moved to XSTORE	Number Deleted	Scan Resume Point (TCMSTFRM)
MDC XSTORE Steal Complete	HCPFTV HCPFMV	4107	Target Size for XSTORE Cache	Actual Size of XSTORE Cache (TCMXSTOR)	Number Deleted	Address of Last TCHBK Done (TCMSTXTC)	TCMFLAG| TCMFLAG2|0| Loop Count
MDC I/O Retry as Non-STD	HCPFTI HCPFMI	4108	VDEV Address	Hash Key TCHKEY	Hash Table Entry TCHBK Address	SAVFLAG| SAVFLAG2|00	SAVBK Address (ESA/390) SVGBK Address (z/Arch)
Minidisk Cache Control Unit Simulator Entry	HCPFTS	4110	VMDBK Address	Virtual Device number| Hash ID for Minidisk	IORBK Address	FCTBK Address	Last Seek CCW CCHH arg. or Last FBA Locate CCW Block arg. in chan. prog.

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Minidisk Cache Control Unit Simulator Exit (Abort)	HCPFTE	4111	VMDBK Address	Virtual Device number| Hash ID for Minidisk	IORBK Address	FCTBK Address	FCTFLAG| FCTSTYPE| FCTFLAG2|00
Minidisk Cache Control Unit Simulator Exit (Successful)	HCPFTS	4112	VMDBK Address	Virtual Device number| Hash ID for Minidisk	IORBK Address	FCTBK Address	FCTFLAG| FCTSTYPE| FCTFLAG2|00
Minidisk Cache Track Purge for Diagnose or *BLOCKIO	HCPBIU	4115	R11 VMDBK Address	Virtual Device|00 00 VDEV-DEV	Track Access Key (DVTFTKEY)	TCHBK Address (or 0 if track not accessed) (DVTTCHBK)	Track Format Data (or 0 if track not accessed) (DVTTKDAT)
Minidisk Cache Results from Diagnose or *BLOCKIO	HCPBIM HCPBIR HCPDGB HCPDGD	4116	R11 VMDBK Address	Virtual Device|Hash ID VDEV-DEV (DBCHSID)	DBCSWTCH| DBCSWTC2| DBCflg[7]|DBCSWTC4	Block size[6]|00| DBCRC	Address list (or 0)
Start Interpretive Execution Assist CSCH, CC=0	HCPPTI	5000	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist CSCH, CC=3	HCPPTI	5003	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist HSCH, CC=0	HCPPTI	5010	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist HSCH, CC=1	HCPPTI	5011	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist HSCH, CC=2	HCPPTI	5012	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist HSCH, CC=3	HCPPTI	5013	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist MSCH, CC=0	HCPPTI	5020	RDEVDEV|RDEVSUB	Path Management Control Words (PMCW Words 1-3)			PMCW Word 6
Start Interpretive Execution Assist MSCH, CC=1	HCPPTI	5021	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Start Interpretive Execution Assist MSCH, CC=2	HCPPTI	5022	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist MSCH, CC=3	HCPPTI	5023	RDEVDEV|RDEVSUB	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist SSCH, CC=0	HCPPTI	5030	RDEVDEV|RDEVSUB	IORBK Address	Operation Request Block (ORB)		
Start Interpretive Execution Assist SSCH, CC=1	HCPPTI	5031	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist SSCH, CC=2	HCPPTI	5032	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist SSCH, CC=3	HCPPTI	5033	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist TSCH, CC=0	HCPPTI	5050	RDEVDEV|RDEVSUB	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Start Interpretive Execution Assist TSCH, CC=1	HCPPTI	5051	RDEVDEV|RDEVSUB	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Start Interpretive Execution Assist TSCH, CC=3	HCPPTI	5053	RDEVDEV|RDEVSUB	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist TPI, CC=0	HCPPTI	5060	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist TPI, CC=1	HCPPTI	5061	00 00 00 00	Subsystem ID	Interrupt Parameter	Interrupt ID Word	00 00 00 00
Start Interpretive Execution Assist RSCH, CC=0	HCPPTI	5080	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist RSCH, CC=1	HCPPTI	5081	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Start Interpretive Execution Assist RSCH, CC=2	HCPPTI	5082	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist RSCH, CC=3	HCPPTI	5083	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist Cancel I/O Request	HCPIOX	50A0	RDEVDEV|RDEVSUB	IORBK Address	Return Code	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist CSCH, CC=0 (type 3)	HCPPTI	5300	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist CSCH, CC=3 (type 3)	HCPPTI	5303	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist HSCH, CC=0 (type 3)	HCPPTI	5310	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist HSCH, CC=1 (type 3)	HCPPTI	5311	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist HSCH, CC=2 (type 3)	HCPPTI	5312	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist HSCH, CC=3 (type 3)	HCPPTI	5313	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist MSCH, CC=0 (type 3)	HCPPTI	5320	E3 F3|RDEVSUB	Path Management Control Words (PMCW Words 1-3)			PMCW Word 6
Start Interpretive Execution Assist MSCH, CC=1 (type 3)	HCPPTI	5321	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist MSCH, CC=2 (type 3)	HCPPTI	5322	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist MSCH, CC=3 (type 3)	HCPPTI	5323	E3 F3|RDEVSUB	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Start Interpretive Execution Assist SSCH, CC=0 (type 3)	HCPPTI	5330	E3 F3|RDEVSUB	IORBK Address	Operation Request Block (ORB)		
Start Interpretive Execution Assist SSCH, CC=1 (type 3)	HCPPTI	5331	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist SSCH, CC=2 (type 3)	HCPPTI	5332	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist SSCH, CC=3 (type 3)	HCPPTI	5333	E3 F3|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist TSCH, CC=0 (type 3)	HCPPTI	5350	E3 F3|RDEVSUB	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Start Interpretive Execution Assist TSCH, CC=1 (type 3)	HCPPTI	5351	E3 F3|RDEVSUB	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Start Interpretive Execution Assist TSCH, CC=3 (type 3)	HCPPTI	5353	E3 F3|RDEVSUB	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist Type 3 Subchannel Cancel I/O Request	HCPIOX	53A0	E3 F3|RDEVSUB	IORBK Address	Return Code	00 00 00 00	00 00 00 00
Start Interpretive Execution Assist Interrupt	HCPIPT	5500	RDEVDEV|RDEVSUB	A(RDEV) RDEV Address	PFXRNUSR	I/O Old PSW	
		5501	See Format 2 table				
Start Interpretive Execution Assist Interrupt (type 3)	HCPIPT	5530	E3 F3|RDEVSUB	A(RDEV) RDEV Address	PFXRNUSR	I/O Old PSW	
		6001	See Format 2 table				
		6003	See Format 2 table				
		6004	See Format 2 table				
		6005	See Format 2 table				
		6006	See Format 2 table				

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
		6010	See Format 2 table				
		6011	See Format 2 table				
		6012	See Format 2 table				
		6013	See Format 2 table				
		6014	See Format 2 table				
		6015	See Format 2 table				
		6016	See Format 2 table				
		6020	See Format 2 table				
		6021	See Format 2 table				
		6022	See Format 2 table				
		6023	See Format 2 table				
		6024	See Format 2 table				
		6025	See Format 2 table				
Check APPC/IUCV Path	HCPBUV	7000	CC	PTHBK Address	MDEBK Address		Caller's Return Address
Locate PTHBK by APPC Path	HCPBUV	7002	CC[4]	PTHBK Address or Path ID	MDEBK Address		Caller's Return Address
Locate PTHBK by Inter-System Communication Facility Session	HCPBUV	7003	CC[4]	PTHBK Address or Session ID	MDEBK Address		Caller's Return Address
Add a Message (MDEBK) to a Work Queue	HCPBUV	7004			MDEBK Address	Queue Anchor	Caller's Return Address
Get a Message (MDEBK) from a Work Queue	HCPBUV	7005	CC		MDEBK Address	Queue Anchor	Caller's Return Address
Queue a Message for a Link	HCPCMS	7008	FUN|FLG| MDEMSGID	PTHBK Address	MDEBK Address	MDECODE| TYP|AFL	LNKBK Address
Dequeue Message for a Link	HCPCMR	7009	FUN|FLG| MDEMSGID	PTHBK Address	MDEBK Address	MDECODE| TYP|AFL	LNKBK Address
Check CP Application Path	HCPBUV	700A	CC	PTHBK Address	MDEBK Address		Caller's Return Address

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Add a Message (CARBK) to a Work Queue	HCPBUV	700B			CARBK Address	Queue Anchor	Caller's Return Address
Get a Message (CARBK) from a Work Queue	HCPBUV	700C	CC		CARBK Address	Queue Anchor	Caller's Return Address
VCTC Define	HCPCTV	7900	X owner VMDBK Address	X VDEV Address	00 00 00 00	00 00 00 00	VDEVDEV|00 00
VCTC Detach	HCPCTV	7901	X owner VMDBK Address	X VDEV Address	Y VDEV Address	CACXPEND| CACXACTV| CACXCNTL| CACXUINT	CACYPEND| CACYACTV| CACYCNTL| CACYUINT
VCTC Couple	HCPCTV	7902	X owner VMDBK Address	X VDEV Address	Y VDEV Address	Target VMDBK Address	Target VDEV Address
VCTC Interface Disconnect	HCPCTV	7903	X owner VMDBK Address	X VDEV Address	Y VDEV Address	CACXPEND| CACXACTV| CACXCNTL| CACXUINT	CACYPEND| CACYACTV| CACYCNTL| CACYUINT
VCTC Selective Reset	HCPCTV	7904	X owner VMDBK Address	X VDEV Address	Y VDEV Address	CACXPEND| CACXACTV| CACXCNTL| CACXUINT	CACYPEND| CACYACTV| CACYCNTL| CACYUINT
VCTC System Reset	HCPCTV	7905	X owner VMDBK Address	X VDEV Address	Y VDEV Address	CACXPEND| CACXACTV| CACXCNTL| CACXUINT	CACYPEND| CACYACTV| CACYCNTL| CACYUINT
VCTC I/O Simulation	HCPCTC	7906	X owner VMDBK Address	X VDEV Address	Y VDEV Address	CACXPEND| CACXACTV| CACXCNTL| CACXUINT	CCW opcode| CCW flags| DataLength
VCTC Unsolicited Interrupt	HCPCTV	7907	X owner VMDBK Address	X VDEV Address	Y VDEV Address	CACXPEND| CACXACTV| CACXCNTL| CACXUINT	VDEVDEV|00| UnitStatus
Clear Logical Subchannel, CC=0	HCPIOS	8000	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Clear Logical Subchannel, CC=3	HCPIOS	8003	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Clear Logical Subchannel, CC=0	HCPIOS	8010	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Halt Logical Subchannel, CC=1	HCPIOS	8011	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Halt Logical Subchannel, CC=3	HCPIOS	8013	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Modify Logical Subchannel, CC=0	HCPIOS	8020	RDEVDEV|RDEVSUB	Path Management Control Words (PMCW) 1-3			PMCW Word 6
Modify Logical Subchannel, CC=1	HCPIOS	8021	RDEVDEV|RDEVSUB	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Modify Logical Subchannel, CC=3	HCPIOS	8023	RDEVDEV|RDEVSUB	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Start Logical Subchannel, CC=0	HCPIOS	8030	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	Operation Request Block (ORB)		
Start Logical Subchannel, CC=1	HCPIOS	8031	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Logical Subchannel, CC=3	HCPIOS	8033	RDEVDEV|RDEVSUB	IORBK Address	Operation Request Block (ORB)		
Start Logical Subchannel, CC=0 Sense	HCPIFI	8038	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	Operation Request Block (ORB)		
Start Logical Subchannel, CC=1 Sense	HCPIFI	8039	RDEVDEV|RDEVSUB	IORBK Address	00 00 00 00	00 00 00 00	00 00 00 00
Start Logical Subchannel, CC=3 Sense	HCPIFI	803B	RDEVDEV|RDEVSUB	IORBK Address	Operation Request Block (ORB)		
Test Logical Subchannel, CC=0	HCPIFI	8050	RDEVDEV|RDEVSUB	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Test Logical Subchannel, CC=1	HCPIFI	8051	RDEVDEV|RDEVSUB	Subchannel Status Word (SCSW)			Extended Status Word (ESW)
Test Logical Subchannel, CC=3	HCPIFI	8053	RDEVDEV|RDEVSUB	Active IORBK Address (RDEVAIOR)	00 00 00 00	00 00 00 00	00 00 00 00
Logical I/O Sense Data Received	HCPIFI	80F0	RDEVDEV|IORSCNT	Sense Data Bytes 0-15			
		80F1–84FF	See Format 2 table				

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Logical I/O Interrupt	HCPIFI	8500	RDEVDEV|RDEVSUB	Real Device Block Address	PFXRNUSR	I/O Old PSW	
		8501	See Format 2 table				
		8620	See Format 2 table				
		8630	See Format 2 table				
		8720	See Format 2 table				
		8730	See Format 2 table				
		8A00	See Format 2 table				
		9A00	See Format 2 table				
		9A11	See Format 2 table				
SIGP Instruction	HCPSGP	AE00	Processor from Address	Processor to Address	SIGP Order	Program Mask (CC)	SIGP
		B500	See Format 2 table				
		B504	See Format 2 table				
		C010	See Format 2 table				
		C011	See Format 2 table				
VM Service Diag Trace	*8	F000	*8	*8	*8	*8	*8
VM Service Diag Trace	*8	F001	*8	*8	*8	*8	*8
VM Service Diag Trace	*8	F002	*8	*8	*8	*8	*8
VM Service Diag Trace	*8	F003	*8	*8	*8	*8	*8
VM Service Diag Trace	*8	F004	*8	*8	*8	*8	*8
VM Service Diag Trace	*8	F005	*8	*8	*8	*8	*8
VM Service Diag Trace	*8	F006	*8	*8	*8	*8	*8
VM Service Diag Trace	*8	F007	*8	*8	*8	*8	*8
VM Service Diag Trace	*8	F008	*8	*8	*8	*8	*8

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
VM Service Diag Trace	*[8]	F009	*[8]	*[8]	*[8]	*[8]	*[8]
Processor is Check Stopped	HCPMCH	F400	00 00 00 00	MCIC		00 00 00 00	
Check Stop Processor Recovery	HCPMCH	F401	Address of Failed Processor (STAP Format)		Machine Check Interrupt Code (MCIC)	Address of Failed Processor's Prefix Page	00 00 00 00
Trace Page Full During Machine Check Handling	HCPMCH	F40F	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Free Storage Extend Replenishment	HCPALF HCPPTE HCPPTF	F702	High half of virtual Frame Address	Low half of virtual Frame Address	31–bit real Frame Address	31–bit address of owning virtual systems's VMDBK	Number of free storage extend frames needed
		F704	See Format 2 table				
		F706	See Format 2 table				
		F708	See Format 2 table				
		F70A	See Format 2 table				
Snapdump invoked via Command	HCPSNP	F800	C' xxx'-- Where xxx is ABEND Module ID	C'nnn '-- Where nnn is ABEND Number	VMDBK Address	SAVBK Address (GPR13)	00 00 00 00
Snapdump invoked via System error	HCPSNP	F801	C' xxx'-- Where xxx is ABEND Module ID	C'nnn '-- Where nnn is ABEND Number	VMDBK Address	SAVBK Address (GPR13)	80 00 00 00[5] or 40 00 00 00
Snapdump Exit	HCPSNP	F802	C' xxx'-- Where xxx is ABEND Module ID just processed	C'nnn '-- Where nnn is ABEND Number	VMDBK Address	SAVBK Address (GPR13) or 00 00 00 00	00 00 00 00, 80 00 00 00[5], or 40 00 00 00
Call Exit Start	HCPZXU	F900	0000xxxx Exit number	xxxxxxxx Address of XITBK for this exit	xxxxxxxx R1, addr of plist for this exit	xxxxxxxx R14, real address of caller	xxxxxxxx XITCALLS value
Call Exit Routine Start	HCPZXU	F910	0000xxxx Exit number	xxxxxxxx Address of the XCRBK	xxxxxxxx R1, addr of plist for this exit	xxxxxxxx R14, real address of caller	xxxxxxxx XCRATMPT value
Call Exit Routine Finish	HCPZXU	F920	0000xxxx Exit number	yyyyxxxx Both halves of the return code	xxxxxxxx R1, addr of plist for this exit	xxxxxxxx R14, real address of caller	xxxxxxxx XCRATMPT value
Call Exit Finish	HCPZXU	F930	0000xxxx Exit number	0000xxxx Return code sent back to mainframe	xxxxxxxx R1, addr of plist for this exit	xxxxxxxx R14, real address of caller	xxxxxxxx XITCALLS value
Call Exit Routine Finish (NOP)	HCPZXU	F940	0000xxxx Exit number	yyyyxxxx Value is always 00000000	xxxxxxxx R1, addr of plist for this exit	xxxxxxxx R14, real address of caller	xxxxxxxx XCRATMPT value
Clear Parser plist	HCPZPR	FA00	Address of plist to clear	Length of plist to clear	VMDBK Address	00 00 00 00	Caller's Return address

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4
OFFSET (hex)		**A**	**C**	**10**	**14**	**18**	**1C**
Resume Trace After Soft Abend	HCPABN	FFEE	TOD Clock (Bits 0-31)	TOD Clock (Bits 32-64)	VMDBK Address	C'+SNAP+'	
Time Stamp Trace Entry	HCPTTI	FFFE	00 00 00 00	00 00 00 00	00 00 00 00	1st word of TOD clock value	2nd word of TOD clock value
Suspend Trace During Soft Abend	HCPABN	FFFF	C'xxx'-- Where xxx is MODE ID	C'nnn'-- Where nnn is Abend Number	VMDBK Address	C'aaaaaaaa'-- Where aaaaaaaa is the Name of Run User at Time of Soft Abend	

Notes:

1. These fields are generated only by HCPPAH and HCPPAU. HCPVOD, HCPVOS, and HCPVIR generate zeros (00 00 00 00).
2. CPSYSCD is filled in for non-APPC paths, Flags2 is filled in for APPC paths.
3. IOR|TRQSCHED - Bit seven in this field indicates whether this field contains the address of the TRQBK (the bit is on) or the address of the IORBK (the bit is off).
4. If CC equals 0, the PTHBK Address is traced. If CC does not equal 0, the PTHBK could not be located and the path ID requested is traced.
5. If a system error invoked a soft ABEND, which was set to SNAPDUMP by the SET ABEND command, a value of 80 00 00 00 will be present. A value of 40 00 00 00 will be present if invoked by the HCPABEND macro with SNAPDUMP as the defined ABEND type.
6. For Diagnose X'18' results, this is the DBCMAXSZ block size.
7. This flag byte is dependent on the operation type as defined by DBCSWTC2.
   - When DBCSWTC2 bit 5 is on (X'04'), this field contains DBCDOPER
   - When DBCSWTC2 bit 6 is on (X'02'), this field contains DBCBOPER
   - When DBCSWTC2 bit 7 is on (X'01'), this field contains DBCA4SWT
   - When none of the previous bits are on, this field contains DBC18SWT.
8. This field varies based on the temporary diagnostic code given to the customer by the VM Service organization.

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4	CONTENTS OF REG N+5
OFFSET (hex)		**E**	**10**	**18**	**20**	**28**	**30**	**38**
I/O Interruption	HCPIFI	0501	RDEV DEV RDEV SUB M1 M2 M3 M4	00 00 00 00 A(RDEV)	00 00 00 00 RDEVAIOR	00 00 00 00 RDEVNXTI	00 00 00 00 RDEVERPQ	0
Virtual XA I/O Interruption	HCPVOS	0C01	VDEV DEV VDEV SUB 00 00 00 00	SID PARM	00 00 00 00 VDEVPIOR	00 00 00 00 VDEVSIOR	00 00 00 00 VDEVUIOR	00 00 00 00 VDEVIORQ
C Function Return with Error Codes	(Various)	2CA0	Savearea (SZVSB)	Function Header Address	Caller Real Address	Return Code	Reason Code	Error ID
Demand Scan Pass 1	HCPALD	4003	Bits 0-31: count of frames on <2 GB available list Bits 32-63: count of frames on >2 GB available list	Bits 0-31: <2 GB high threshold Bits 32-63: >2 GB high threshold	Bits 0-31: <2 GB low threshold Bits 32-63: >2 GB low threshold	Bits 0-31: number of outstanding <2 GB frame requests Bits 32-63: number of outstanding any frame requests	Bits 0-31: number of steal writes for <2 GB frames Bits 32-63: number of steal writes for >2 GB frames	0
Demand Scan Pass 2	HCPALD	4004	Bits 0-31: count of frames on <2 GB available list Bits 32-63: count of frames on >2 GB available list	Bits 0-31: <2 GB high threshold Bits 32-63: >2 GB high threshold	Bits 0-31: <2 GB low threshold Bits 32-63: >2 GB low threshold	Bits 0-31: number of outstanding <2 GB frame requests Bits 32-63: number of outstanding any frame requests	Bits 0-31: number of steal writes for <2 GB frames Bits 32-63: number of steal writes for >2 GB frames	0
Address Translation	HCPHTR	4020	64–bit Input Virtual Address	64–bit Output Host Address	00 00 00 00 ASTE origin + capability + alt prefix	00 00 00 00 VMDBK Address	Caller's Address	ASCBK Address
I/O Pass Thru Interruption	HCPIPT	5501	RDEV DEV RDEV SUB 00 00 00 00	00 00 00 00 A(RDEV)	00 00 00 00 RDEVAIOR	00 00 00 00 A(VDEV)	0	0
Container Platform Specific Lock	HCPSZK	6001	64-bit Handle Address	64-bit LOCBK Address	00 00 00 00 LOCLOCK	LOCTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT
Container Platform Specific Unlock	HCPSZK	6003	64-bit Handle Address	64-bit LOCBK Address	00 00 00 00 LOCLOCK	LOCTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT
Container Platform Specific Test Lock	HCPSZK	6004	64-bit Handle Address	64-bit LOCBK Address	00 00 00 00 LOCLOCK	LOCTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT
Container Platform Specific Allocate Lock	HCPSZK	6005	64-bit Handle Address	64-bit LOCBK Address	00 00 00 00 LOCLOCK	LOCTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT
Container Platform Specific Deallocate Lock	HCPSZK	6006	64-bit Handle Address	64-bit LOCBK Address	00 00 00 00 LOCLOCK	LOCTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4	CONTENTS OF REG N+5
OFFSET (hex)		**E**	**10**	**18**	**20**	**28**	**30**	**38**
Container Platform Specific Latch Entry	HCPSZL	6010	64-bit Handle Address	64-bit LATBK Address	LATSTATE LATQUEUE	LATTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT
Container Platform Specific Latch Exit	HCPSZL	6011	64-bit Handle Address	64-bit LATBK Address	LATSTATE LATQUEUE	LATTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT
Container Platform Specific Unlatch Entry	HCPSZL	6012	64-bit Handle Address	64-bit LATBK Address	LATSTATE LATQUEUE	LATTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT
Container Platform Specific Unlatch Exit	HCPSZL	6013	64-bit Handle Address	64-bit LATBK Address	LATSTATE LATQUEUE	LATTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT
Container Platform Specific Test Latch	HCPSZL	6014	64-bit Handle Address	64-bit LATBK Address	LATSTATE LATQUEUE	LATTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT
Container Platform Specific Allocate Latch	HCPSZL	6015	64-bit Handle Address	64-bit LATBK Address	LATSTATE LATQUEUE	LATTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT
Container Platform Specific Deallocate Latch	HCPSZL	6016	64-bit Handle Address	64-bit LATBK Address	LATSTATE LATQUEUE	LATTHRED	00 00 00 00 PFXNOLOC	00 00 00 00 PFXSLCNT
Container Platform Specific Wait Event Entry	HCPSZE	6020	64-bit Handle Address	64-bit EVNBK Address	00 00 00 00 EVNQUEUE	00 00 00 00 EVNLATLK	0	0
Container Platform Specific Wait Event Exit	HCPSZE	6021	64-bit Handle Address	64-bit EVNBK Address	00 00 00 00 EVNQUEUE	00 00 00 00 EVNLATLK	0	0
Container Platform Specific Notify Event Entry	HCPSZE	6022	64-bit Handle Address	64-bit EVNBK Address	00 00 00 00 EVNQUEUE	00 00 00 00 EVNLATLK	0	0
Container Platform Specific Notify Event Exit	HCPSZE	6023	64-bit Handle Address	64-bit EVNBK Address	00 00 00 00 EVNQUEUE	00 00 00 00 EVNLATLK	0	0
Container Platform Specific Allocate Event	HCPSZE	6024	64-bit Handle Address	64-bit EVNBK Address	00 00 00 00 EVNQUEUE	00 00 00 00 EVNLATLK	0	0
Container Platform Specific Deallocate Event	HCPSZE	6025	64-bit Handle Address	64-bit EVNBK Address	00 00 00 00 EVNQUEUE	00 00 00 00 EVNLATLK	0	0

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4	CONTENTS OF REG N+5
OFFSET (hex)		**E**	**10**	**18**	**20**	**28**	**30**	**38**
External Interruption (z/Arch)	HCPEXT	8100	C ' EXT' 00 00 00 00	External Interruption Parameter, CPU Address, External Interruption Code	External Old PSW		0	0
Program Interrupt (z/Arch)	HCPPRG	8300	ILC Interruption code VMDBK address	Fault Address	Program Old PSW		Data Exception code, Monitor class number, PER code, ATMID	PER address
Program Interrupt (z/Arch)	HCPIFI	8501	RDEV DEV RDEV SUB 00 00 00 00	00 00 00 00 A(RDEV)	00 00 00 00 RDEVAIOR	0	0	0
Obtain Host Logical Aligned Free Storage	HCPAFS	8620	Block's ID X'<XXX'	Double Words Requested	Host Logical Assigned Block Address	Absolute Assigned Block Address	Requestor VMDBK Address	Caller's VMDBK Address
Obtain Absolute Aligned Free Storage	HCPAFR	8630	00000000	Double Words Requested	Absolute Assigned Block Address	Requestor VMDBK Address	Caller's Return Address	00000000
Return Host Logical Aligned Free Storage	HCPAFS	8720	00000000	Double Words Returned	Host Logical Returned Block Address	Absolute Returned Block Address	Caller's VMDBK Address	Caller's Return Address
Return Absolute Aligned Free Storage	HCPAFR	8730	00000000	Double Words Returned	Absolute Returned Block Address	Caller's VMDBK Address	Caller's Return Address	00000000
Run User (z/Arch)	HCPRUN	8A00	VMDMMODE C'RUN' SIEINTWD	00 00 00 00 VMDBK address	Guest PSW (SIEGPSW)		0	0
Run User in Virtual Sie Mode (z/Arch)	HCPWRU	9A00	00 C'WRU' RGuest VMDBK address	00 00 00 00 vSIE VMDBK address	VGuest PSW (SIEGPSW)		RGuest SIEBK.0–3 (V,S,MX,M)	Shadow CR1 (VMDWSGCR1)
Virtual Sie Interception (z/Arch)	HCPWRU	9A11	00 C'WRUNI' SIEICODE, SIEICFLG	00 00 SIEIPARM (= SIEINST, —IPB)	VGuest PSW (SIEGPSW)		VGuest SIEBK.0–3 (V,S,MX,M)	0
Interception, Not Instruction	HCPRUN	B500	VMDMMODE, 00 00 SIEICODE SIEINTWD	00 00 00 00 VMDBK address	Guest PSW (SIEGPSW)		0	0

NAME	MODULE	TRACE ID	CONTENTS OF REG N	CONTENTS OF REG N+1	CONTENTS OF REG N+2	CONTENTS OF REG N+3	CONTENTS OF REG N+4	CONTENTS OF REG N+5
OFFSET (hex)		**E**	**10**	**18**	**20**	**28**	**30**	**38**
Guest Instruction Interception (z/Arch)	HCPPRV	B504	00 00 00 00 Low-order four bytes of Guest GPR1	00 00 00 00 VMDBK address	Guest PSW (SIEGPSW)		VMDMMODE SIEICFLG SIEINST SIEIPB	0
Virtual Machine Lock Storage	HCPVMS	C010	Caller's Address	VMDBK Address	Virtual Address To Be Locked	Real Frame Address	0	0
Virtual Machine Unlock Storage	HCPVMS	C011	Caller's Address	VMDBK Address	Virtual Address To Be Unlocked	Real Frame Address	0	0
Return Free Storage Backed Pages	HCPSXP	F704	SXS Logical Page Address	SXSSTATEG Contents	Absolute Frame Address	FRMSTATEG Contents	RSASXUFG or RSASXUFS	Caller's Return Address
Obtain Free Storage Backed Pages	HCPSXP	F706	SXS Logical Page Address	SXSSTATEG Contents	Absolute Frame Address	FRMSTATEG Contents	RSAXTEND (count of pages missing from free storage reserved page list)	Caller's Return Address
Return Free Storage Frame	HCPPTE HCPPTF	F708	Host Logical FRMTE Address	FRMCSWRD	Real Frame Address	0	0	0
Obtain Free Storage Frame	HCPPTE	F70A	0	0	0	Real Frame Address	0	0

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, New York 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs

and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, New York 12601-5400
U.S.A.
Attention: Information Request

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of International Business Machines corporation in the United States, other countries, or both:

- BookMaster
- DFSMS/VM
- eServer
- IBM
- IBMLink
- Language Environment
- MVS
- NetView
- OpenExtensions
- Performance Toolkit for VM
- RACF
- SP
- System/390
- VSE/ESA
- VTAM
- z/Architecture
- z/OS
- z/VM
- zSeries

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

For a list of z/VM terms and their definitions, see the *z/VM: Glossary* book.

The glossary is also available through the online HELP Facility. For example, to display the definition of “cms”, enter:

```
help glossary cms
```

You will enter the glossary HELP file and the definition of “cms” will be displayed as the current line. While you are in the glossary HELP file, you can also search for other terms.

If you are unfamiliar with the HELP Facility, you can enter:

```
help
```

to display the main HELP menu, or enter:

```
help cms help
```

for information about the HELP command.

For more information about the HELP Facility, see the *z/VM: CMS User’s Guide*.

Bibliography

This bibliography lists the books in the z/VM product library. For abstracts of these books and information about current editions and available media, see *z/VM: General Information*.

Where to Get z/VM Books

z/VM books are available from the following sources:

- IBM Publications Center at www.ibm.com/shop/publications/order/
- z/VM Internet Library at www.ibm.com/eserver/zseries/zvm/library/
- *IBM Online Library: z/VM Collection*, SK2T-2067
- *IBM Online Library: z/VM Collection on DVD*, SK5T-7054

z/VM Base Library

The following books describe the facilities included in the z/VM base product.

Overview

z/VM: General Information, GC24-6095

z/VM: Glossary, GC24-6097

z/VM: License Information, GC24-6102

Installation, Migration, and Service

z/VM: Guide for Automated Installation and Service, GC24-6099

z/VM: Migration Guide, GC24-6103

z/VM: Service Guide, GC24-6117

z/VM: VMSES/E Introduction and Reference, GC24-6130

Planning and Administration

z/VM: CMS File Pool Planning, Administration, and Operation, SC24-6074

z/VM: CMS Planning and Administration, SC24-6078

z/VM: Connectivity, SC24-6080

z/VM: CP Planning and Administration, SC24-6083

z/VM: Getting Started with Linux on System z9 and zSeries, SC24-6096

z/VM: Group Control System, SC24-6098

z/VM: I/O Configuration, SC24-6100

z/VM: Running Guest Operating Systems, SC24-6115

z/VM: Saved Segments Planning and Administration, SC24-6116

z/VM: TCP/IP Planning and Customization, SC24-6125

eServer zSeries 900: Planning for the Open Systems Adapter-2 Feature, GA22-7477

System z9 and eServer zSeries: Open Systems Adapter-Express Customer's Guide and Reference, SA22-7935

System z9 and eServer zSeries: Open Systems Adapter-Express Integrated Console Controller User's Guide, SA22-7990

z/OS and z/VM: Hardware Configuration Manager User's Guide, SC33-7989

Customization and Tuning

z/VM: CP Exit Customization, SC24-6082

z/VM: Performance, SC24-6109

Operation

z/VM: System Operation, SC24-6121

z/VM: Virtual Machine Operation, SC24-6128

Application Programming

z/VM: CMS Application Development Guide, SC24-6069

z/VM: CMS Application Development Guide for Assembler, SC24-6070

z/VM: CMS Application Multitasking, SC24-6071

z/VM: CMS Callable Services Reference, SC24-6072

z/VM: CMS Macros and Functions Reference, SC24-6075

z/VM: CP Programming Services, SC24-6084

z/VM: CPI Communications User's Guide, SC24-6085

z/VM: Enterprise Systems Architecture/Extended Configuration Principles of Operation, SC24-6094

z/VM: Language Environment User's Guide, SC24-6101

z/VM: OpenExtensions Advanced Application Programming Tools, SC24-6104

z/VM: OpenExtensions Callable Services Reference, SC24-6105

z/VM: OpenExtensions Commands Reference, SC24-6106

z/VM: OpenExtensions POSIX Conformance Document, GC24-6107

z/VM: OpenExtensions User's Guide, SC24-6108

z/VM: Program Management Binder for CMS, SC24-6110

z/VM: Reusable Server Kernel Programmer's Guide and Reference, SC24-6112

z/VM: REXX/VM Reference, SC24-6113

z/VM: REXX/VM User's Guide, SC24-6114

z/VM: Systems Management Application Programming, SC24-6122

z/VM: TCP/IP Programmer's Reference, SC24-6126

Common Programming Interface Communications Reference, SC26-4399

Common Programming Interface Resource Recovery Reference, SC31-6821

z/OS: Language Environment Concepts Guide, SA22-7567

z/OS: Language Environment Debugging Guide, GA22-7560

z/OS: Language Environment Programming Guide, SA22-7561

z/OS: Language Environment Programming Reference, SA22-7562

z/OS: Language Environment Run-Time Messages, SA22-7566

z/OS: Language Environment Writing ILC Applications, SA22-7563

z/OS MVS Program Management: Advanced Facilities, SA22-7644

z/OS MVS Program Management: User's Guide and Reference, SA22-7643

End Use

z/VM: CMS Commands and Utilities Reference, SC24-6073

z/VM: CMS Pipelines Reference, SC24-6076

z/VM: CMS Pipelines User's Guide, SC24-6077

z/VM: CMS Primer, SC24-6137

z/VM: CMS User's Guide, SC24-6079

z/VM: CP Commands and Utilities Reference, SC24-6081

z/VM: Quick Reference, SC24-6111

z/VM: TCP/IP User's Guide, SC24-6127

z/VM: XEDIT Commands and Macros Reference, SC24-6131

z/VM: XEDIT User's Guide, SC24-6132

CMS/TSO Pipelines Author's Edition, SL26-0018

System Diagnosis

z/VM: CMS and REXX/VM Messages and Codes, GC24-6118

z/VM: CP Messages and Codes, GC24-6119

z/VM: Diagnosis Guide, GC24-6092

z/VM: Dump Viewing Facility, GC24-6093

z/VM: Other Components Messages and Codes, GC24-6120

z/VM: TCP/IP Diagnosis Guide, GC24-6123

z/VM: TCP/IP Messages and Codes, GC24-6124

z/VM: VM Dump Tool, GC24-6129

z/OS and z/VM: Hardware Configuration Definition Messages, SC33-7986

Books for z/VM Optional Features

The following books describe the optional features of z/VM.

Data Facility Storage Management Subsystem for VM

z/VM: DFSMS/VM Customization, SC24-6086

z/VM: DFSMS/VM Diagnosis Guide, GC24-6087

z/VM: DFSMS/VM Messages and Codes, GC24-6088

z/VM: DFSMS/VM Planning Guide, SC24-6089

z/VM: DFSMS/VM Removable Media Services, SC24-6090

z/VM: DFSMS/VM Storage Administration, SC24-6091

Directory Maintenance Facility

z/VM: Directory Maintenance Facility Commands Reference, SC24-6133

z/VM: Directory Maintenance Facility Messages, GC24-6134

z/VM: Directory Maintenance Facility Tailoring and Administration Guide, SC24-6135

Performance Toolkit for VM™

z/VM: Performance Toolkit, SC24-6136

Resource Access Control Facility

External Security Interface (RACROUTE) Macro Reference for MVS and VM, GC28-1366

Resource Access Control Facility: Auditor's Guide, SC28-1342

Resource Access Control Facility: Command Language Reference, SC28-0733

Resource Access Control Facility: Diagnosis Guide, GY28-1016

Resource Access Control Facility: General Information, GC28-0722

Resource Access Control Facility: General User's Guide, SC28-1341

Resource Access Control Facility: Macros and Interfaces, SC28-1345

Resource Access Control Facility: Messages and Codes, SC38-1014

Resource Access Control Facility: Migration and Planning, GC23-3054

Resource Access Control Facility: Security Administrator's Guide, SC28-1340

Resource Access Control Facility: System Programmer's Guide, SC28-1343

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Readers' Comments — We'd Like to Hear from You

z/VM
Diagnosis Guide
version 5 release 2

Publication No. GC24-6092-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

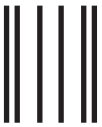
Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
GC24-6092-01

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, New York 12601-5400

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Program Number: 5741-A05

Printed in USA

GC24-6092-01

Spine information:

z/VM Diagnosis Guide version 5 release 2