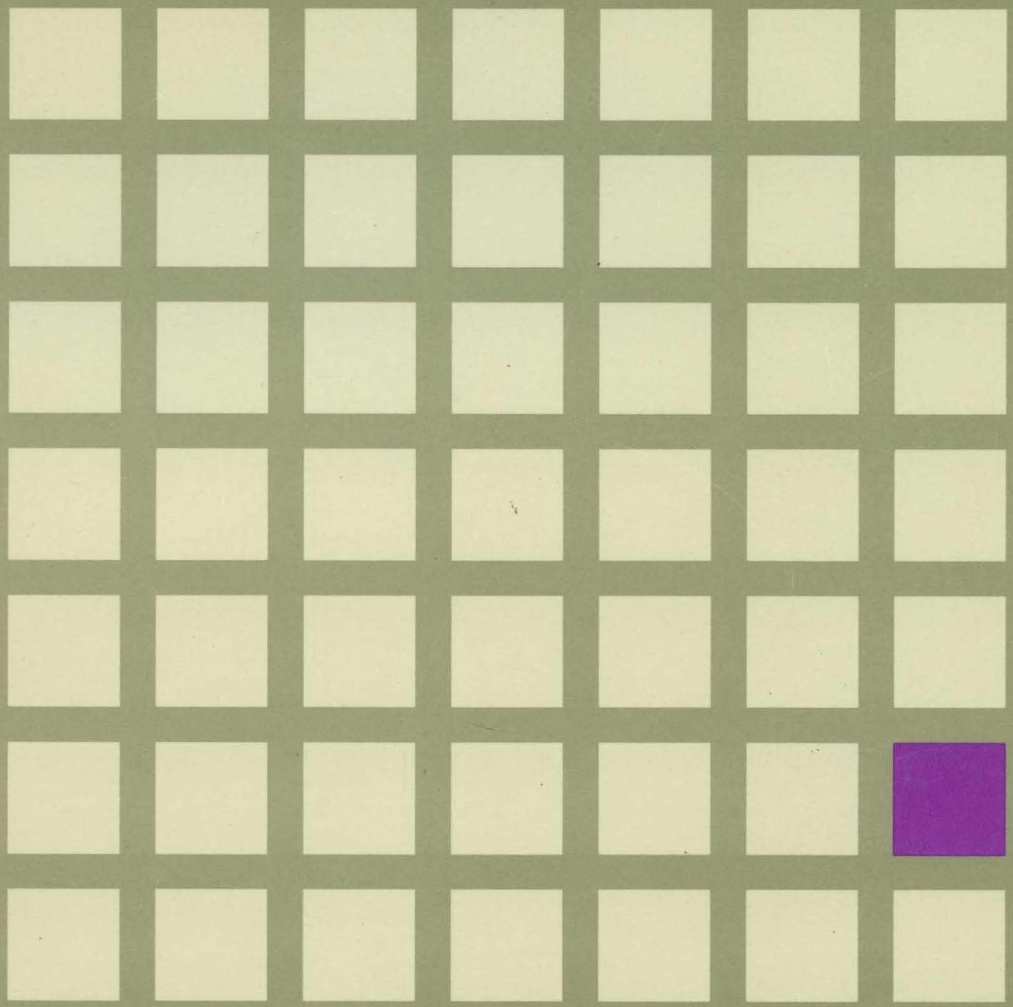
LY24-5241-01

Virtual Machine/System Product

Diagnosis Guide

Release 6



LY24-5241-01

Virtual Machine/System Product

Diagnosis Guide

Release 6

Second Edition (July 1988)

This edition, LY24-5241-01, is a major revision of LY24-5241-00 and applies to the Virtual Machine/System Product Release 6, program number 5664-167, and to all subsequent releases of this product until otherwise indicated in new editions or Technical Newsletters. It contains material formerly included in the *VM/SP Problem Reporting Guide* (discontinued after Release 5), and the *VM/SP GCS Diagnosis Reference* (discontinued after Release 5). Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the *IBM System/370, 30xx, 4300, and 9370 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Summary of Changes

For a list of changes, see "Summary of Changes" on page 265.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Ordering Publications

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. Publications are *not* stocked at the address given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Dept. G60, P.O. Box 6, Endicott, NY, U.S.A. 13760. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

The form for readers' comments provided at the back of this publication may also be used to comment on the VM/SP online HELP facility.

Preface

This is a guide to identify, report, solve, and collect information about problems in the Virtual Machine/System Product (VM/SP), program number 5664-167. It is intended for system programmers, system analysts, and others who know assembler language and have experience with programming concepts and techniques.

This manual is one of many reference manuals for VM/SP or VM/SP HPO system programmers. Other books include:

- *VM/SP CP Diagnosis Reference*
- *VM/SP CMS Diagnosis Reference*
- *VM/SP Interactive Problem Control System Guide and Reference*
- *VM/SP CMS Shared File System Administration*
- *VM/SP Connectivity Programming Guide and Reference*
- *VM/SP Connectivity Planning, Administration, and Operation.*

This *VM/SP Diagnosis Guide* consists of:

- **Chapter 1. Introduction to Debugging**, which contains an overview of the debugging environment.
- **Chapter 2. Debugging the Virtual Machine**, which contains a description of commands used to display or dump data, set and query system features, trace events, and alter storage.
- **Chapter 3. Debugging CP**, which contains a description of commands and macros used to debug CP problems.
- **Chapter 4. Debugging CMS**, which contains a description of commands used to debug CMS.
- **Chapter 5. Debugging the SFS Server Machine**, containing a description of methods used to collect information to debug the SFS server machine.
- **Chapter 6. Debugging GCS**, which contains a description of tracing and dumping facilities used to debug GCS problems.
- **Chapter 7. Debugging TSAF**, which contains a description of methods used to collect information to debug TSAF.
- **Chapter 8. Debugging AVS**, which contains a description of methods used to collect information to debug AVS.
- **Appendix A. Problem-Specific Checklists**, which contains a checklists of information you should gather before calling IBM for help with abends, loops, wait states, and other problems.
- **Appendix B. Control Registers**, which contains a description of the control register allocation and assignments.
- **Appendix C. Stand-Alone Dump Formats**, which contains a description of the stand-alone dump tape format, DASD format, and printer format.

- **Appendix D. GCS Control Blocks**, which contains the layouts of some GCS control blocks and key fields that are used for identifying problems in a VM/SNA environment.
- **Summary of Changes**, which contains a summary of the enhancements made to this manual since the last edition was issued.
- **Glossary of Terms and Abbreviations**, which explains or defines the terms, acronyms, and abbreviations that appear in this manual.
- **Bibliography**, which lists prerequisite and corequisite publications.
- **Index**, which lists the topics in this manual alphabetically and points to the pages where they are discussed.

Contents

Chapter 1. Introduction to Debugging

VM/SP 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 an IBM System/370. Therefore, the person trying to determine the cause of a VM/SP 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, GCS, TSAF, or AVS
- The problem program, which executes under the control of a virtual machine operating system.

Refer to:

- Chapter 2, "Debugging the Virtual Machine" on page 47 for information on how to debug problems within a virtual machine, and "Commands that Trace Events in Virtual Machines" on page 54 for information on how to debug application programs.
- Chapter 3, "Debugging CP" on page 71 for information on CP.
- Chapter 4, "Debugging CMS" on page 131 for information on CMS.
- Chapter 5, "Debugging the SFS Server Machine" on page 145 for information on the SFS (Shared File System) Server Machine.
- Chapter 6, "Debugging GCS" on page 155 for information on GCS (the Group Control System.
- Chapter 7, "Debugging TSAF" on page 227 for information on TSAF (the Transparent Services Access Facility).
- Chapter 8, "Debugging AVS" on page 235 for information on AVS (APPC/VM VTAM Support).

For information explaining how to use Interactive Problem Control System (IPCS) for debugging, refer to the *VM/SP Interactive Problem Control System Guide and Reference*.

If a problem is caused by a guest operating system, refer to 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 VM/370 or VM/SP, refer to the *VM/SP Installation Guide* for information on applying PTFs.

How To Start Debugging

A good approach to debugging is:

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.

When running VM/SP, you must also decide whether the problem is in CP, the virtual machine, or the problem program.

Does a Problem Exist?

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

- Abend (abnormal end)
- Unexpected or incorrect results
- Loop
- Wait state.

Abend: The most obvious indication of a problem is the abnormal termination (abend) of a program. An abend occurs when an error condition that cannot be resolved by the system causes a program to terminate prematurely. Whenever a program abnormally terminates, a message is issued. This message provides information that can help you isolate the problem.

Unexpected or Incorrect Results: Another obvious indication of a problem is unexpected or incorrect output or results. If your output is missing, incorrect, or in a different format than expected, some problem exists.

Infinite Loops: A loop is a set of instructions that is executed repeatedly as long as a certain condition is present. Infinite loops are caused when the condition that is supposed to be satisfied in the loop is never reached. If your program takes longer to execute than anticipated, it might be in a loop. If your output is repeated more than expected, your program may be in a loop.

Wait States: A VM/SP 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. Enables wait states occur frequently, and are quite easily resolved or resolve themselves. Disabled wait states are not easily resolved and almost always a sign of a serious problem, but often a message is issued alerting you to a disabled wait. If your program is taking longer than expected to execute, the virtual machine may be in a wait state.

Other problems: 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.

Identifying the Problem

Identifying problems is not always easy. Abnormal termination is sometimes indicated by an error message, and unexpected results become apparent once the output is examined. Looping and wait state conditions may not as easy to identify.

Table 1 on page 13 helps you to identify problem types and the areas where they may occur.

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 instruction. This return code describes to your program the condition that arose when your machine tried to carry out the instruction. 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 VM/SP system—its system programming—is no different. Depending upon the return code received from an instruction in its system software (or, for that matter, in an application program that you are running on VM/SP) your system is programmed to react in a certain way.

The severity of return codes differs. Some conditions are handled more smoothly than others.

For an explanation of the meaning of individual return codes, consult *VM/SP System Messages and Codes*. Many return codes are also recorded in the *VM/SP CMS User's Guide*.

Messages: A *message* is a sentence or phrase transmitted by VM/SP that describes a situation or problem that 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.

VM/SP has thousands of messages and is programmed to generate a particular message when a given situation or problem occurs.

VM/SP messages consist of these parts:

- The message identifier
- The message text.

A *message identifier* is a combination of letters and numbers that uniquely identifies a message. The *message text* is the set of words that indicates your problem. The message text may contain *message variables*, which are spaces filled with important data, making the message more informative.

The message identifier consists of four fields: a prefix, a module code, a message number, and a severity code. The prefix corresponds to the component that issued the message. Here are the message identifier prefixes with their corresponding VM/SP components:

- **DMK** -- CP
- **DMS** -- CMS
- **CSI** -- GCS
- **DMM** -- IPCS
- **ATS** -- TSAF
- **AGW** -- AVS.

The module code indicates which module generated the message. The message number is associated with the condition that caused the message to be generated. The severity code is a letter that indicates what kind of condition caused the message. Not every severity code applies to every component, but the following list contains all the possibilities:

- A -- Immediate action required
- I -- Information message
- R -- Response
- W -- Warning or system wait
- E -- Error
- S -- Severe error
- T -- Terminating error.

You may receive the following message if you fail to issue the CPTRAP command correctly. Use this example to identify the parts of a message.

```
DMKTRP020E Userid missing or invalid
```

Messages can help identify problems, both large and small. Be aware of the messages you receive when your system is experiencing problems. For an explanation of individual messages, consult *VM/SP System Messages and Codes.*

Where to Find Evidence

Depending on the severity of the problem, or abend, you must take action to identify and correct the problem. You may have to use one of the following sources to find exactly where a major problem occurred:

- A Dump
- A Nucleus Load Map (NUCMAP)
- Registers
- The Program Status Word (PSW)
- The Channel Status Word (CSW)
- The Channel Address Word (CAW)
- The Console Log
- A Trace
- The Program Event Recording Facility (PER).

Dumps: A *dump* is a record of the contents of your machine's storage at a given moment. A dump can appear either on-line or printed on paper. You are interested in finding the moment when malfunctions, errors, or problems begin.

Depending upon the type of dump you request and where the dump comes from, it can include the data contained in the following:

- 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 VM/SP system's real storage.
- Control blocks.
- General-purpose and floating-point registers.
- Control registers.
- The time-of-day clock.
- The processor timer.
- The program status word (PSW).

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

- **A CP dump.** This can be a dump of your entire VM/SP system or just the storage directly "owned" by CP.
- **A stand-alone dump.** Sometimes, a problem can be so severe that your system can't even produce a CP dump on its own. So, every VM/SP system is equipped with a special program that will produce a dump of real storage, regardless of how severe the problem is. We call it a "stand-alone" dump because the program that produces it stands alone or independent of the rest of the system programming. Since 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** running in your VM/SP system. For example, you can request a dump of the virtual machine containing CMS, RSCS, or any guest operating system that resides in VM/SP.
- 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.

A dump is useful when dealing with a problem in your VM/SP system. A dump is a picture of the system's (or virtual machine's) memory at the moment of malfunction or error. The problem is likely to be somewhere in the picture.

Dumps are especially helpful in dealing with infinite loops, wait states, and abends. Moreover, the information contained in a NUCMAP complements the information in a dump. When you create a dump, it is wise to obtain a NUCMAP. Refer to the material in the section titled "NUCMAP or Nucleus Load Map." See also "Creating a Dump" on page 20 .

NUCMAP or Nucleus Load Map: A *nucleus load map* (NUCMAP) is a record that contains the following information.

- A list of the storage addresses of all control sections (CSECTs). A *control section* or CSECT is that 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 present in main 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.

---

[1] These activities are performed by the system programmer or system operator using the MAINT virtual machine. This is the virtual machine that is used to install, service, and maintain your VM/SP system. The *VM/SP Installation Guide* explains these activities.

One NUCMAP exists for CP, another for CMS, and another for GCS. NUCMAPs are often called load maps, particularly by the Interactive Problem Control System (IPCS).

VM/SP creates a NUCMAP each time CP or CMS is built—that is, when your system is first installed or after it is repaired or modified.[1] So, the NUCMAPs are kept up-to-date.

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

NUCMAPs 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.
- The GCSNUC MAP is stored on the GCS virtual disk at address 595.

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

General purpose registers contain numeric or alphabetic values being manipulated by the user program currently running. Floating-point registers are used to 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 execution 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 VM/SP system.

Your machine uses a register to store a piece of data that it is using *right now*. 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. Depending on the problem, a great deal can be learned by examining the contents of your system's registers if a malfunction, error, or problem occurs.

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

Program Status Word: The *program status word* (PSW) is an area in storage that indicates your system's general status. The PSW is a doubleword (or 64 bits) in storage that is divided into several fields. Concentrate on these fields:

Bit 6 Indicates whether your system will accept *(or is enabled for)* input interrupts. If this bit is set to 0, your machine is not enabled for input. If this bit is set to 1, your machine will accept input interrupts.

Bit 14 Indicates whether your VM/SP system is in a wait state. If this bit is set to 0, your system is not in the wait state, and execution 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 indicates whether the wait state is enabled (1) or disabled (0).

Normally, your system indicates that it is in a wait state by either displaying the word "WAIT" on the system display terminal or by activating the system console's WAIT light.

Bits 40-63 Contains the address of the next instruction your machine is set to execute.

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 determine the contents of the PSW by using the CP DISPLAY command with the PSW option. You can also determine the PSW by looking at a dump.

Channel Status Word: A channel is a device that manages and directs the flow of data between your VM/SP system's main storage and a storage device (printer, DASD, terminal, or tape). A *channel status word* (CSW) is a doubleword of storage describing the condition of a storage device and the channel to which it is attached.

The CSW contains fields that indicate, among other things:

- The general status of the channel. For example, is the channel idle or busy?
- The general status of the input/output device. For example, is the device operating normally or has some problem occurred? If there is a problem, what kind is it? What is the input/output interrupt status of the device?
- The condition under which the last input or output operation was completed.
- The type of input or output operation underway.
- Whether conditions have developed that prevent the normal flow of data through the channel.

The information in a CSW is usually complete once the I/O interrupt associated with the operation in question has occurred. This information can be helpful in tracking down problems involving unexpected output results and input/output errors.

You can determine the contents of the CSW by using the CP DISPLAY command with the CSW option. You can also determine the CSW by looking at a dump.

Channel Address Word: The *channel address word* (CAW) is a fullword in your VM/SP system's storage that contains the address of the channel program as well as control information for that program.

A *channel program* is a special program that manages the operation of a channel.

The information in a CAW can be helpful in tracking down problems involving unexpected output results and input/output errors. The channel program itself can be located and examined for possible errors.

You can determine the contents of the CAW by using the CP DISPLAY command with the CAW option. You can also determine the CAW by looking at a dump.

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

When problems arise in the system, we 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.
- All commands and instructions that the operator has entered at his console.
- All 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*.

But 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, then 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 issue the following command to begin the recording. The best place for this command is in the profile for the virtual machine in question. That way, you know a console log is always being recorded. Or, you can always issue the command from the command line and have it in effect temporarily, such as,

```
cp spool console start
```

Issue

```
cp close console
```

to create a console log of the information recorded up to this point and puts 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.

Traces: A *trace* is a chronological record, usually printed, of every "major event" that has taken place within your VM/SP system or within a virtual machine running there. Each "major event" corresponds to a program or a set of instructions that your system or virtual machine has executed, each 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.

VM/SP automatically maintains what is called the CP internal trace table. This table is a record of all events that have taken place in CP, and is described in the section titled "CP Internal Trace Table" on page 74.

An internal trace table is also maintained for GCS. Consult the "Internal Tracing Facilities" on page 157 for more information.

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

TRACE A CP command that traces general virtual machine activity. This command records trace data at a terminal, a virtual printer, or both. For more information, check the *VM/SP CP General User Command Reference*.

PER A CP command that monitors events in a virtual machine. The PER command monitors such events as instruction fetching, successful branching, or a change in a register or storage address. For more information, check the *VM/SP CP General User Command Reference* and *VM/SP CP System Command Reference*. Also, review the sections of this book titled "The Program Event Recording Facility (PER)" on page 11 and "Using the CP PER Command" on page 59.

CPTRAP A CP command and facility that stores certain diagnostic information in a virtual reader file. This information includes the entries in the CP internal trace table plus other data gathered from CP and various virtual machines. For more information, check "Debugging with the CPTRAP Facility" on page 95 and the *VM/SP CP System Command Reference*.

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, check "Using the ITRACE Command and GTRACE Macro" on page 158 and the *VM/SP Group Control System Command and Macro Reference*.

ETRACE A GCS command that actually records a given event in the GCS internal trace table. The ETRACE command works closely with the CPTRAP command. For more information, check Chapter 6, "Debugging GCS" on page 155 and the *VM/SP Group Control System Command and Macro Reference*.

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 a SNA device. Among those items you can trace in a SNA environment are:

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

Detailed information on all of this is available in the *VTAM Diagnosis Guide* and the *VTAM Diagnosis Reference*.

Symptom Records: Your VM/SP system contains a component called the Interactive Problem Control System (IPCS). Fundamentally, IPCS helps you report, diagnose, and manage any problems you have with VM/SP.

Anyone who is involved in VM/SP problem-solving should be familiar with IPCS. For details, refer to the *VM/SP Interactive Problem Control System Guide and Reference*.

For now, it is useful to review one IPCS concept—the *symptom record*. A symptom record is a collection of data conveying basic information about some VM/SP software failure. It includes the following:

- The component, release, and service level of your VM/SP system. These items identify your VM/SP system *software* to the IBM Support Center.
- The model number and serial number of your particular processor. These items identify your system *hardware* to the IBM Support Center.
- The date and time of day that the symptom record was created.
- The type of dump of which the symptom record has become a part.

  The symptom record is located within the first two kilobytes of any dump that is produced.
- The error code associated with the problem or error at hand.
- The ID of the failing component—for example, **5749DMK00**. (This particular ID refers to CP.)
- The name of the program module within the failing component wherein the error or problem occurred.
- The contents of all the registers.
- The *register PSW difference*. This designates which register might recently have been used as a base register. This information helps locate where the error is likely to be in storage.

The importance of this information goes beyond merely conveying information about a single problem. The IBM Support Center also uses this information to determine whether a certain problem is related to one that has already been reported. If it is, then you and IBM know more about the problem than you did before. Moreover time, money, and effort are conserved. Obviously, if PROBLEM X and PROBLEM Y are related, it would be wasteful to treat them as though they were separate and unrelated. Since these relationships are not always immediately obvious, we look to the symptom record.

The Program Event Recording Facility (PER): PER helps you monitor three important events that take place in a virtual machine.

- **Instruction fetch.** Whenever you ask your system to execute an instruction, it has to locate the program needed to execute it. We call this action an *instruction fetch*. In a problem situation, it may be helpful to know what instruction was fetched at a certain moment. If it's the wrong one, you may have found the error.
- **Successful branch.** A program branches when it executes an instruction other than the next sequential instruction. PER helps you trace when and to where a branch occurs. If you suspect that a problem is being caused by a branch to the wrong place or at the wrong time, PER can help.
- **Storage alteration.** As we've said, the contents of areas of storage change frequently in a computer. PER helps you monitor the contents of any register or address at any time. In a problem situation, you can monitor the value of

any key register or area of memory to make sure that it is correct. If it isn't, then you might have found the problem.

Use the PER command to invoke the PER facility. The PER command has many options that allow you to specify exactly what event you want to monitor. Other options let you specify what you want to happen once the event has occurred—that is, do you want the event traced, do you want a dump, or do you want the virtual machine to wait. Still other PER options allow you to define the event very narrowly to avoid having to handle an unnecessarily large amount of data.

For detailed information on the PER command, check the *VM/SP CP General User Command Reference* and *VM/SP CP System Command Reference*. Also refer to "Using the CP PER Command" on page 59.

Analyzing the Problem

Once the type of problem is identified, its cause must be determined. 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 the cause of the problem is not found after the recommended debugging procedures are followed, it may be necessary to undertake the tedious job of desk-checking.

The section "How To Use VM/SP Facilities To Debug" on page 20 describes procedures to follow in determining the cause of various problems that can occur in CP or in the virtual machine. See "Commands that Trace Events in Virtual Machines" on page 54 for information on using VM/SP facilities to debug a problem program.

Figure 1 on page 14, Figure 2 on page 15, and Figure 3 on page 16 summarize the debugging process from identifying the problem to finding the cause.

Table 1. VM/SP Problem Types

Problem Type	Where Abend Occurs	Distinguishing Characteristics
Abend	CP abend CMS abend GCS abend TSAF abend AVS abend	For a complete discussion of reasons for abends and system programmer's actions, see the CP, CMS, GCS, TSAF, and AVS abend codes charts in *VM/SP System Messages and Codes.*
	Virtual machine abend (other than CMS)	When OS or DOS abnormally terminates on a virtual machine, the messages issued and the dumps taken are the same as they would be if OS or DOS abnormally terminated on a real machine. CP may terminate or reset a virtual machine if a unrecoverable channel check or machine check occurs in that virtual machine. The system operator will receive a message at the processor console. Also, the virtual user will be notified by a message that his virtual machine was terminated or reset.
Unexpected Results	CP Virtual machine	If an operating system, other than CMS, executes properly on a real machine, but not properly with CP, a problem exists. Inaccurate data on files, such as spool files, is an error. If a program executes properly under the control of a particular operating system on a real machine, but does not execute 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 *VM/SP System Messages and 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 execute 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. In CMS, the continued typing of the blip characters indicates that processing time is elapsing. 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.

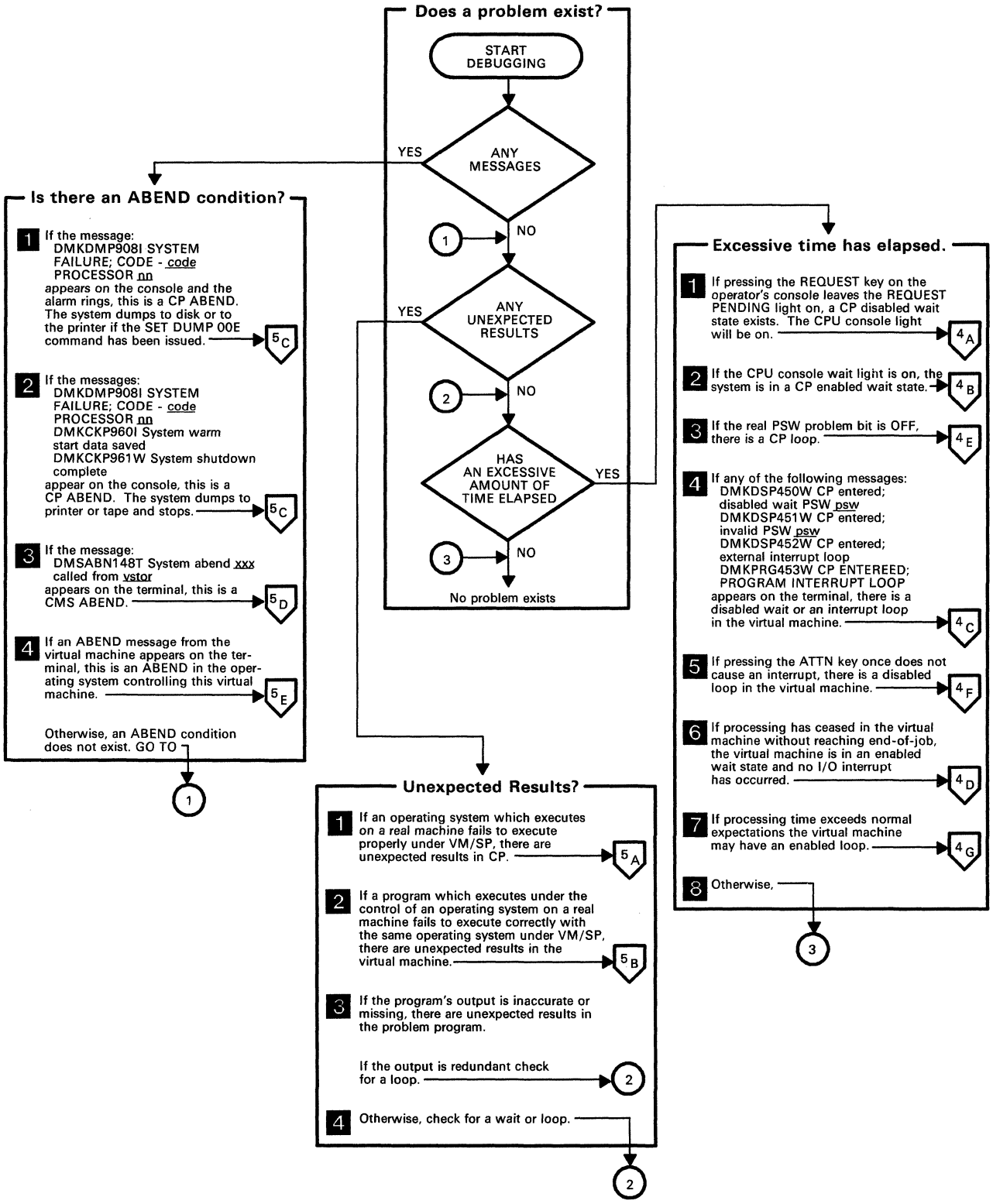


Figure 1. Does a Problem Exist?

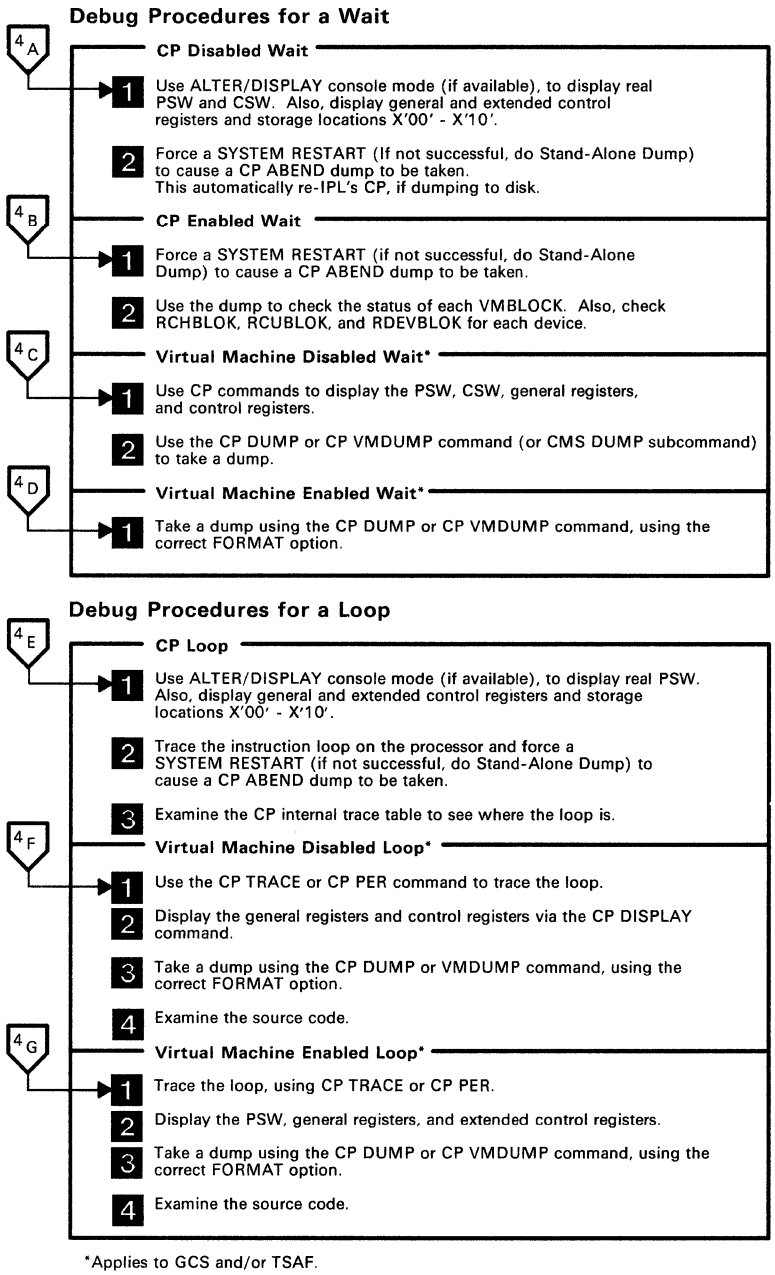


Figure 2. Debug Procedures for Waits and Loops

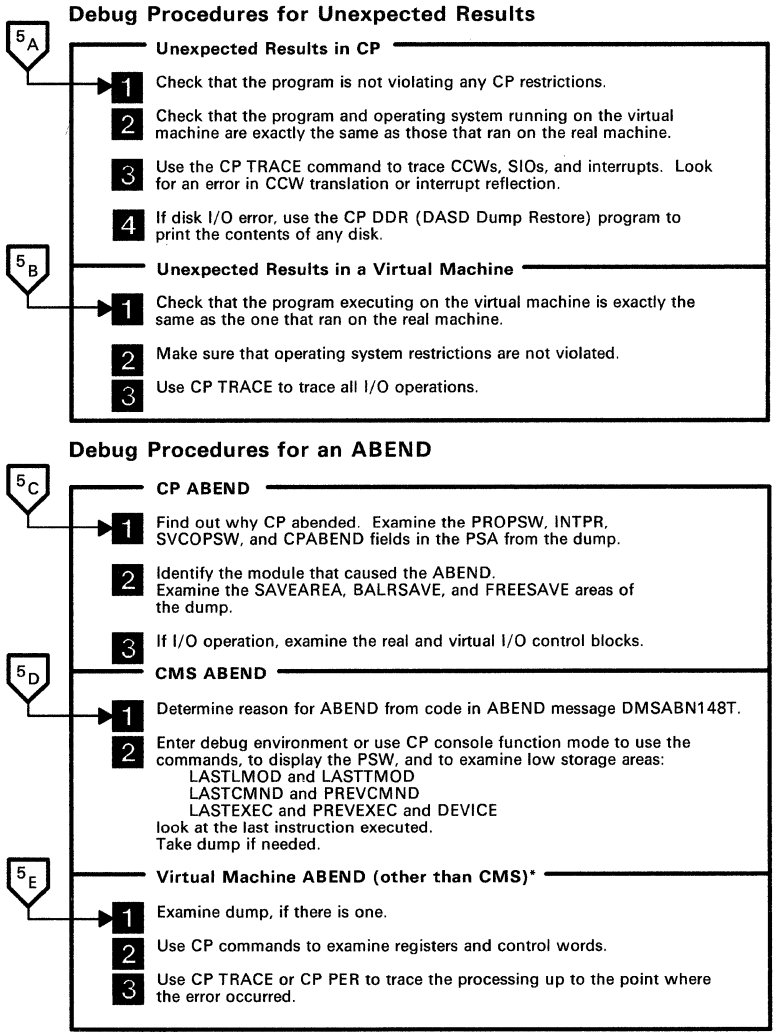


Figure 3. Debug Procedures for Unexpected Results and an Abend

Data Needed Before Calling IBM for Assistance

Note: This section contains general information for all VM/SP-based operating systems.

If you should need to call your IBM Support Center for assistance, it is very important for you to have the following information:

- Problem Inquiry Data Sheet
- List of all applied maintenance for the module(s) involved
- Operator's console log
- Verification that all known errors against the PUT have been applied
- NUCMAP for the failing system.

Problem Inquiry Data Sheet

The Problem Inquiry Data Sheet must be accurately filled-in to ensure that you get the correct solution from IBM. It might be a good idea to make copies of the Problem Inquiry Data Sheet (see Figure 4 on page 19), to have blank sheets available in case you have to call IBM.

System Information: When completing the Problem Inquiry Data Sheet, the QUERY CPLEVEL command should be used to help you to determine:

- Operating system
- Release level
- Service level

of your system.

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

```
query cplevel
```

you could get something that looked like this:

```
VM/SP RELEASE 5, SERVICE LEVEL 501
GENERATED AT 03/01/86 11:33:48 EDT
IPL AT 03/14/86 16:01:31 EDT
```

Make sure that you record system information (first output line from the query cplevel command) 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 could get something that looked like this:

```
CPUID = FF13066043810000
```

This is the 16-digit processor identification associated with the virtual machine. Ignore the FF. The ten digits that follow the FF are the CPU Serial (first six digits representing the processor identification number and the next four digits representing the processor model number). Ignore the last four digits of this 16-digit field.

Note: The system release level, service level, and CPU serial number could also be obtained via IPCS from the Problem Record or through the SYMPTOM subcommand of IPCSSCAN if a dump was created for the problem. See the *VM/SP Interactive Problem Control System Guide and Reference* for more information about IPCSSCAN and SYMPTOM.

Data Sheet Fields: The Problem Inquiry Data Sheet consists of the following fields:

Customer
Enter your business' name.

Date
Enter today's date.

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

Access Code
Enter the customer number that the IBM marketing representative gave 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 which 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 suspected component where problem exists. (For example, CP, CMS, TSAF, etc.)

Problem/Inquiry Description
Enter reason for calling the IBM Support Center.

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

Documentation Available
Indicate 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 APAR number assigned to problem (if defect related).

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

Other
Enter other pertinent information to this problem.

Sheet 1 of ___

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

Operating System, Service Level, and Release Level:
(Output of 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 ___	PUT Level ___
Storage Map ___	Diagnostic Output ___	Service Level ___
Test Data ___	TP CONFIG List(s) ___	VMLOAD List ___

Problem Tracking:

Date	Name	Activity

Resolution PUT Tape
APAR # _____ PTF # _____ Other _____

Figure 4. Problem Inquiry Data Sheet. Use this sheet to collect pertinent data before calling IBM.

How To Use VM/SP Facilities To Debug

Once the problem and the area where it occurs are identified, 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, you have to use the operator's console to display the PSW, but for a virtual machine loop you can display the PSW via the CP DISPLAY command.

The following sections describe specific debugging procedures for the various error conditions. The procedures tell you what to do and what debug tool to use. For example, the procedure may say dump storage using the CP DUMP command. The procedure does not tell you how to use the debug tool. Refer to "Summary of VM/SP Debugging Commands" on page 39 and "Debugging Commands" on page 132 for a description of the debugging tools and commands available to you.

Before examining some of the concrete problems you might encounter in your VM/SP system, review the following procedures that may help you when your system is malfunctioning:

- Creating a Dump
- Locating the CP Internal Trace Table in a Dump.

Creating a Dump

The places a dump can originate in a VM/SP system are:

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

Obtaining a Copy of a CP Restart Dump

Whenever CP abends, it automatically creates a dump. Unless told to do otherwise, your system sends the dump to the virtual reader in the operator's virtual machine.[2] You can then place the dump on a read/write file mode of the operator and send it to a printer.

As an alternative, you can specify in advance other destinations for the CP restart dump. Use the SET DUMP command to indicate where you prefer these dumps be sent whenever one is generated. You can specify a printer, a tape, or a DASD device. The *VM/SP CP System Command Reference* describes the SET DUMP command in detail.

If the SET DUMP command specifies that the dump be sent to a printer, then all you must do is locate the printer and remove the print-out.

[2] The person to receive the dump can also be specified in the DMKSYS module when your system is first installed. Use the SYSDUMP operand of the SYSOPER macro instruction to do this.

If the SET DUMP command specifies that the dump be sent to a tape device, be certain that the appropriate reel of tape is mounted. Then, check to be sure that the tape device on which the reel is mounted is attached to your system. Remember the VM/SP restriction that requires the entire dump to fit on a single reel of tape. Next, issue the following command to identify the printer file to your system. The name you must give to the file is INMOVE.

```
FILEDEF INMOVE PRINTER (RECFM FM LRECL 131)
```

Next, issue the following command to identify to your system the tape file containing the dump. The name you must give to this file is OUTMOVE.

```
FILEDEF OUTMOVE TAP1 (DEN 1600 RECFM U LRECL 132)
```

Finally, print the dump using the following command

```
MOVEFILE OUTMOVE INMOVE
```

The printer should begin producing the dump shortly.

However, if the SET DUMP AUTO command is in effect, thereby specifying that the dump be placed in CP spool space, then you must use the Interactive Problem Control System (IPCS) to process the dump, place it on a read/write file mode, and print it.

Use the IPCSDUMP command to process the dump. Then, use the IPCSPRT command to print the dump. Consult the *VM/SP Interactive Problem Control System Guide and Reference* for detailed information on using these instructions.

Obtaining a Copy of a Stand-Alone Dump

We introduced the concept of a stand-alone dump in the section titled "Dumps" on page 5. Obtaining a stand-alone dump, while not difficult, is a bit too involved to detail here. Refer to "Stand-Alone Dump Facility" on page 123 for details of this procedure.

Obtaining a Copy of a Virtual Machine Dump

Before 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, VS1, 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 VM/SP dump commands. Review the manuals pertaining to the operating system in question.

If a virtual machine contains a VM/SP product or component (such as RSCS or GCS), there are several commands you can use to create a dump.

As you know, when an abend occurs in CP, a dump is produced automatically. But there may be times when you want a dump of the system when one is not produced automatically. You might choose the CP VMDUMP command, which produces a virtual storage dump of any virtual machine running in VM/SP (including a GCS virtual machine like RSCS). The *VM/SP CP General User Command Reference* describes the VMDUMP command in detail. Note that the VMDUMP command is an authorized command. This means that only authorized users can issue it.

Just like CP, GCS produces a dump automatically whenever an abend occurs. And, as with CP, you may want a dump of GCS when one is not produced automatically. You might decide on the GDUMP command, a general use GCS command. Check

the *VM/SP Group Control System Command and Macro Reference* for a details of this command.

The dump created by either the VMDUMP or the GDUMP command may be viewed using IPCS.

The CMS Debugging Facility can help you obtain a dump of a virtual machine running in CMS. Review the material on the CMS DEBUG command in the *VM/SP CMS Command Reference* and the *VM/SP CMS User's Guide.*

Obtaining a Copy of a Communication Controller Dump

If a model 3704 or 3705 communication controller fails or operates erratically, the contents of its storage should be dumped and examined. If you previously issued the NETWORK AUTO command, your system produces this dump automatically when it discovers trouble.

If this automatic feature is not enabled and you decide that you need a dump of a communication controller, perform the activities that follow, using the virtual machine authorized to use the NCPDUMP command.[3]

First, create the dump file with the following command. Substitute the real hexadecimal address of the communication controller for *raddr*. When you issue the NETWORK command, the control program in the communication controller is destroyed. You must reload this program later.

```
NETWORK DUMP raddr
```

Next, print the dump by issuing the following command:

```
NCPDUMP
```

Finally, reload the control program of the communication controller with the following command. Substitute the real hexadecimal address of the communication controller for *raddr*; substitute the name of the control program image you want reloaded into the device for *ncpname*.

```
NETWORK LOAD raddr ncpname
```

For more information on obtaining a storage dump from a communication controller, refer to the *VM/SP Operator's Guide.*

Locating the CP Internal Trace Table in a Dump

The CP internal trace table is the place where CP keeps a detailed record 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.

The CP internal trace table contains information on such things as interrupts, input/output activity, storage allocation, and many of other things. The section "CP Internal Trace Table" on page 74 offers detailed information on the contents of the CP internal trace table.

---

[3] This virtual machine was identified in the DMKSYS module when your system was first installed. The identification is made with the SYSDUMP operand of the SYSOPER macro instruction.

The address of the CP internal trace table is stored at address X'0C'. Each event and its effect are recorded in a 16-byte event record. When the entire table is filled, CP "wraps around" to the top of the table and begins recording there, overlaying what was recorded previously.

Address X'10' contains the address of the byte *following* the end of the trace table. Address X'14' contains the address of the 16-byte area where the next event record will be recorded. To find the address of the last recorded event record, merely subtract X'10' from the contents of address X'14'. When you view a dump using IPCS, there is no need to go to the trouble of finding all these addresses. IPCS gives you a TRACE subcommand that formats each trace table entry for you. Refer to the *VM/SP Interactive Problem Control System Guide and Reference.*

A copy of the CP internal trace table always accompanies a CP restart dump. This is the dump that CP automatically produces whenever it terminates and then restarts itself. For that matter, CP produces this dump when you manually restart the system.

Abend

The following types of abnormal terminations (abend) can occur in VM/SP:

- CP
- CMS
- 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. Table 2 on page 24 lists the possible abend messages and identifies the type of abend for these messages.

Table 2 (Page 1 of 3). Abend Messages

Message	Type of Abend	Description
(Alarm rings) DMKDMP908I SYSTEM FAILURE; CODE - *code* PROCESSOR *nn*	CP	CP abend, system dumps to DASD, printer, or tape. Restart is automatic.
DMKCKP900W SYSTEM RECOVERY FAILURE; PROGRAM CHECK DMKCKP901W SYSTEM RECOVERY FAILURE; MACHINE CHECK DMKCKP902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR NUCLEUS AREA DMKCKP902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR WARM AREA DMKCKH910W SYSTEM RECOVERY FAILURE; INVALID WARM START AREA DMKCKH911W SYSTEM RECOVERY FAILURE; WARM START AREA FULL DMKCKF922W System recovery failure; invalid spooling data	CP	If the checkpoint program encounters a program check, a machine check, a fatal I/O error, or an error relating to a certain warm start area or warm start data conditions, a message is issued indicating the error and CP enters the wait state with code 007 in the PSW.
DMKCKT903W System recovery failure; volid *volid* allocation area cylinder *cylinder* DMKCKT903W System recovery failure; volid *volid* allocation area page *page* DMKCKT912W System recovery failure; volid *volid* not mounted DMKCKV912W System recovery failure; volid *volid* not mounted DMKCKS915E Permanent I/O error on checkpoint area DMKCKT916E Error allocating spool file buffers DMKCKV916E Error allocating spool file buffers DMKCKV917E Checkpoint area invalid; clear storage and cold start	CP	If the checkpoint start program encounters a severe error, a message is issued indicating the error and CP enters the wait state with code 00E in the PSW.
DMKWRM903W System recovery failure; volid *volid* allocation error cylinder *cylinder* DMKWRM903W System recovery failure; volid *volid* allocation error page *page* DMKWRM904W System recovery failure; invalid warm start data DMKWRM912W System recovery failure; volid *volid* not mounted DMKWRM920W No warm start data; checkpoint start for retry DMKWRM921W System recovery failure; unrecoverable I/O error	CP	If the warm start program encounters a severe error, a message is issued indicating the error and CP enters the wait state with code 009 in the PSW.

Table 2 (Page 2 of 3). Abend Messages

Message	Type of Abend	Description
DMKDMP908I SYSTEM FAILURE; CODE - *code* PROCESSOR *nn* DMKCKP960I System warm start data saved DMKCKP961W System shutdown complete	CP	CP abend, system dumps to DASD, printer, or tape. The system stops; the operator must IPL the system to start again.
Optional Messages: DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK DMKDMP906W SYSTEM DUMP FAILURE; MACHINE CHECK DMKDMP907W SYSTEM DUMP FAILURE; FATAL I/O ERROR	CP	If the dump program encounters a program check, a machine check, or a fatal I/O error, a message is issued indicating the error. CP enters the wait state with code 003 in the PSW. If the dump cannot find a defined dump device and if no printer is defined for the dump, CP enters a disabled wait state with code 004 in the PSW.
DMKMCH610W MACHINE CHECK; SUPERVISOR DAMAGE *cpuid* DMKMCT610W MACHINE CHECK; SUPERVISOR DAMAGE *cpuid*	CP	The machine check handler encountered an unrecoverable error with CP.
DMKMCH611W MACHINE CHECK; SYSTEM INTEGRITY LOST *cpuid* DMKMCT611W MACHINE CHECK; SYSTEM INTEGRITY LOST *cpuid*	CP	The machine check handler encountered an error that cannot be diagnosed; system integrity, at this point, is not reliable.
DMKMCH612W MACHINE CHECK TIMING FACILITIES DAMAGE	CP	An error has occurred in the timing facilities. Probable hardware error.
DMKMCT620I MACHINE CHECK; ATTACHED PROCESSOR NOT BEING USED	CP	A malfunction alert, clock error or instruction processing error occurred on the attached processor. The system continues to run in uniprocessor mode.
DMKMCH622W MACHINE CHECK; MULTIPLE CHANNEL ERRORS DMKACR622W MACHINE CHECK; MULTIPLE CHANNEL ERRORS	CP	On a 303x processor, an error affecting one or more channels in a channel group has occurred. CP enters a disabled wait state with code 001 in the PSW.
DMKCCH603W CHANNEL ERROR DMKACR603W CHANNEL ERROR	CP	There was a channel check condition from which the channel check handler could not recover. CP enters the wait state with code 002 in the PSW.
DMKMCH622W MACHINE CHECK; MULTIPLE CHANNEL ERRORS	CP	There was a group error machine check from which the machine check handler could not recover. CP enters a wait state with code 001 in the PSW.

Table 2 (Page 3 of 3). Abend Messages

Message	Type of Abend	Description

DMSABN148T System abend *xxx* called from *vstor*	CMS	CMS abend, system will accept commands from the terminal. Enter:
		`#CP VMDUMP 0-END FORMAT CMS DSS`
		to create a dump spool file in your reader. Re-IPL CMS and then enter:
		`IPCSDUMP`
		to format the dump.

DMS5FE3034E File pool server system error occurred - *modulename nn*	SFS Server	An internal error occurred within the file pool server. A dump is taken according to the dump option chosen in the start-up parameters. This is a system error. (If the file pool server ends abnormally, a mini-dump is displayed.)
CSIABD232E Abend *xxx-yyy* occurred during abend ESTAE processing	GCS	During the ESTAE abend processing an abend occurred.
CSIABD233E Abend *xxx-yyy* occurred during abend TASKEXIT processing	GCS	During the TASKEXIT abend processing an abend occurred.
CSIABD234E Abend *xxx-yyy* occurred during abend Resource Manager processing	GCS	Specified ABEND occurred while processing a GCS resource. Termination processing completed but all resources may not be cleaned up. Check dump for more details.
CSIABD235E Abend *xxx-yyy* occurred during abend internal processing	GCS	Specified ABEND occurred during the processing of another ABEND. Termination processing did not complete. Check dump for more details.
Others Refer to OS and DOS publications for the abnormal termination messages.	Other	When OS or DOS abnormally terminates on a virtual machine, the message issued and the dumps taken are the same as they would be if OS or DOS abnormally terminated on a real machine.

Note: For TSAF or AVS abends, see the *VM/SP System Messages and Codes*. The following descriptions provide guidelines for debugging each type of abend.

CP Abend

When CP abnormally terminates, a dump is taken. This dump can be directed to tape or printer, or dynamically allocated to a direct access storage device. The output device for a CP abend dump is specified by the CP SET DUMP command. See *VM/SP CP System Command Reference* for a description of the SET DUMP command.

Use the dump to determine why CP terminated and then determine how to correct the condition. See "Reading CP Abend Dumps" on page 78 for detailed

information on reading a CP abend dump. You can view the dump interactively, using IPCS.

Reason for the CP Abend: CP will terminate and take an abnormal termination dump under three conditions:

1. Program Check in CP

   Examine the program old PSW (PROPSW) and the program interrupt code (INTPR) fields in the Prefix Storage Area (PSA) to determine the failing module.

2. Module Issuing an SVC 0

   Examine the SVC old PSW (SVCOPSW) and abend code (CPABEND) fields in the PSA to determine the module that issued the SVC 0 and the reason it was issued.

   CPABEND contains an abnormal termination code. The first three characters identify the failing module (for example, abend code TRC001 indicates DMKTRC is the failing module).

3. Operator forcing a CP system restart on Processor Console

   Examine the restart old PSW (RSRTOPSW) field in the PSA to find the location of the instruction that was executing 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.)

Note: The same conditions that cause an abnormal termination on a uniprocessor configuration cause an abnormal termination on an attached processor.

Examine Low Storage Areas: The information in low storage specifies the status of the system at the time CP terminated. Status information is stored in the PSA. You should be able to tell the module that was executing by looking at the PSA. Refer to the appropriate save area (SAVEAREA, BALRSAVE, or FREESAVE) to see how that module started to execute. The PSA is described in *VM/SP CP Data Areas and Control Blocks*.

Examine the real and virtual control blocks to find the status of I/O operations. Figure 6 on page 84 shows the relationship of CP control blocks.

Examine the CP internal trace table. This table can be extremely helpful in determining the events that preceded the abend. See "CP Internal Trace Table" on page 74 for a description of how to use the trace table.

If you are using IPCS to view the dump, you can use the TRACE subcommand. For details, see the *VM/SP Interactive Problem Control System Guide and Reference*.

The values in the general purpose registers can help you to locate the current IOBLOK and VMBLOK and the save area. Refer to "Reading CP Abend Dumps" on page 78 for detailed information on the contents of the general purpose registers.

If the program old PSW (PROPSW) or the SVC old PSW (SVCOPSW) points to an address beyond the end of the resident nucleus, the module that caused the abend is a pageable module. Refer to "Reading CP Abend Dumps" on page 78 to find out how to identify that pageable module. Use the CP load map that was created when

the VM/SP system was generated to find the address of the end of the resident nucleus.

CMS Abend

When CMS abnormally terminates, any abend exit routines established via the ABNEXIT macro 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:

```
DMSABN148T System abend xxx called from vstor
```

where *xxx* is the abend code and *vstor* is the address of the instruction causing the abend. The DMSABN module issues this message. Then, CMS waits for a command to be entered from the terminal.

Because CMS is an interactive system, you will probably want to use its debug facilities to examine status. You may be able to determine the cause of the abend without taking a dump.

The debug program is located in the resident nucleus of CMS and has its own save and work areas. The debug program itself does not alter the status of the system knowing that routines and data cannot be overlaid unless you specifically request it. Likewise, you can use the CP commands in debugging knowing that you cannot inadvertently overlay storage because the CP and CMS storage areas are completely separate.

Reason for the CMS Abend: First determine the reason CMS abnormally terminated. There are four types of CMS abnormal terminations:

1. Program Exception

   Control is given to the DMSITP (CMS Interrupt Handler) routine whenever a hardware program exception occurs. If no SPIE or STAE exits have been specified, DMSITP issues the message:

   ```
   DMSITP141T exception exception occurred at vstor in
              routine routine
   ```

   and invokes DMSABN (the abend routine). The possible programming exceptions and related abend codes are listed in the *VM/SP System Messages and Codes*, under 141T.

2. ABEND Macro

   Control is given to the DMSSAB routine whenever a user routine executes the ABEND macro. The abend code specified in the ABEND macro appears in the abnormal termination message DMSABN155T.

3. Halt Execution command (HX)

   Whenever the virtual machine operator signals attention and types "HX," CMS terminates and types "CMS."

4. System Abend

   A CMS system routine can abnormally terminate by issuing the DMSABN macro. The first three hexadecimal digits of the system abend code appear in the CMS abend message, DMSABN148T. The format of the DMSABN macro is:

[*label*]	**DMSABN**	{ *code* / *(reg)* } [,**TYPCALL=** [**SVC** / **BALR**]]

label
is any valid Assembler language label.

code
is the abnormal termination code (X'0' through X'FFF') that should appear in the DMSABN148T system termination message.

(reg)
is the register containing the abnormal termination code.

TYPCALL=
specifies how control is passed to the abnormal termination routine, DMSABN. Routines that do not reside in the nucleus should use TYPCALL=SVC to generate CMS SVC 203 linkage. Nucleus-resident routines should specify TYPCALL=BALR so that a direct branch to DMSABN is generated.

If a CMS SVC handler abnormally terminates, that routine can set an abend flag and store an abend code in NUCON (the CMS nucleus constant area). After the SVC handler has finished processing, the abend condition is recognized. The DMSABN abend routine types the abend message, DMSABN148T, with the abend code stored in NUCON.

What to do when CMS Abnormally Terminates: After an abend, two courses of action are available in CMS. As an alternative to the CMS methods, by signalling attention, you can enter the CP command mode and use CP's debugging facilities.

Two courses of action available in CMS are:

1. Issue the DEBUG command and enter the debug environment.

   The most common problem you might encounter is an abnormal termination resulting from a program interruption. When a program running on a CMS virtual machine abnormally terminates (abends), you receive, at your terminal, the message:

```
DMSITP141T exception exception occurred at vstor in routine routine
```

   and your virtual machine is returned to the CMS environment. From the message you can determine the types of program checks (operation, privileged operation, execution, protection, addressing, etc.) and, often, the instruction address in your program at which the error occurred.

   Sometimes this is enough information for you to correct the error in your source program, recompile it and attempt to execute it again.

When this information does not immediately identify the problem in your program, you can begin debugging procedures using VM/SP. To access your program's storage areas and registers you can enter the command:

```
debug
```

immediately after receiving the abend message.

2. Issue a CMS command other than DEBUG, and the abend routine, DMSABN, performs its abend recovery and then passes control to the DMSINT routine to process the command just entered.

The abend recovery function:

1. Clears the console input buffer and program stack.
2. Terminates all VMCF activity.
3. Reinitializes the work area stack for reentrant CMS nucleus modules.
4. Reinitializes the SVC handler, DMSITS, and frees all stacked save areas.
5. Clears the auxiliary directories, if any. Invokes "FINIS * * *," to close all files, and to update the master file directory.
6. Frees storage, if the DMSEXT module is in virtual storage.
7. Zeroes out the MACLIB directory pointers.
8. Frees the CMS work area, if the CMS subset was active.
9. Frets the RLDDATA buffer, used by the CMS loader to retain relocation information for the GENMOD process, if it is still allocated.
10. Issues the STAE, SPIE, TTIMER, and STAX macros to cancel any outstanding OS exit routines. Frees any TXTLIB, MACLIB, or LINK tables.
11. Calls with a purge PLIST, all nucleus extensions that have the "SERVICE" attribute defined.
12. Drops all nucleus extensions that do not have the "SYSTEM" attribute. Also drops any nucleus extensions that are in type user storage.
13. Drops all SUBCOM SCBLOCKS that do not have the "SYSTEM" attribute.
14. Frees console path and device entry control blocks.
15. Drops all storage resident execs that do not have the "SYSTEM" attribute.
16. Clears all immediate commands that are not nucleus extensions with the "SYSTEM" attribute; returns all associated free storage.
17. Calls DMSCLN to zero out the userword of the SRPI command.
18. Calls DMSWITAB to delete all windows and vscreens that do not have the "SYSTEM" attribute.
19. Resets the storage keys for the whole virtual machine, except the nonshared pages, according to FREETAB. Saves the setting for KEYPROTECT.
20. Zeroes out all FCB, DOSCB, and LABSECT pointers.
21. Frees all storage of type user.
22. Restores the setting for KEYPROTECT.
23. Zeroes out all interrupt handler pointers in IOSECT.

24. Turns the SVCTRACE command off.
25. Closes the virtual punch and printer; closes the virtual reader with the HOLD option.
26. Reinitializes the VSE lock table used by CMS/DOS and CMS/VSAM.
27. Zeroes out all OS loader blocks, and frees the FETCH work area.
28. Cleans up the CMS IUCV environment based on the existence of the CMS id block.
29. Clears all ABNEXIT set and returns storage.
30. Computes the amount of system free storage that should be allocated and compares this amount with the amount of free storage actually allocated. Types a message to the user if the two amounts are unequal.
31. Issues a STRINIT and releases any pages remaining in the flush list via a call to DMSPAGFL, if all storage is accounted for.

After abend recovery has completed, control passes to DMSINT at entry point DMSINTAB to process the next command.

When the amount of storage actually allocated is less than the amount that should be allocated, the message

```
DMSABN149T nnn (HEX xxx) doublewords of system storage
           have been destroyed; re-IPL CMS
```

appears on the terminal. If the amount of storage actually allocated is greater than the amount that should be allocated, the message

```
DMSABN150W nnn (HEX xxx) doublewords of system storage
           were not recovered
```

A Debugging Procedure: When a CMS abend occurs, use the CP commands to examine the PSW and specific areas of low storage. You can also use the CMS DEBUG command to examine the PSW and the register contents. For instructions on how to use the CP commands, see "Summary of VM/SP Debugging Commands" on page 39 and "Commands that Trace Events in Virtual Machines" on page 54.

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

1. Display the PSW. (Use the CP DISPLAY command or CMS DEBUG command.) 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. The information in low storage can tell you more about the cause of the abend:

Field	Contents
LASTLMOD	Contains the name of the last module loaded into storage via the LOADMOD command.
LASTTMOD	Contains the name of the last module loaded into the transient area.

LASTCMND Contains the name of the last command issued from the CMS or XEDIT command line. If a command issued in a CMS EXEC abnormally terminates, 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 Contains the name of the next-to-last command issued from the CMS or XEDIT command line. If a command issued in a CMS EXEC abnormally terminates, 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 Contains the name of the last CMS EXEC procedure issued from the CMS or XEDIT command line. EXEC 2 and System Product Interpreter do not update this field.

PREVEXEC Contains the name of the next-to-last CMS EXEC procedure issued from the CMS or XEDIT command line. EXEC 2 and System Product Interpreter do not update this field.

DEVICE Identifies the device that caused the last I/O interrupt. The low storage areas examined depend on the type of abend.

These fields are contained in NUCON. See *VM/SP CMS Data Areas and Control Blocks* for more details of NUCON.

3. Once you have identified the module that caused the abend, examine the specific instruction. Refer to the source code listing if available.
4. If you have not identified the problem at this time, take a dump by issuing the VMDUMP command. Refer to "Reading CMS Abend Dumps" on page 136 for information on reading a CMS dump. If you can reproduce the problem, try the CP or CMS tracing facilities.

SFS Abend

For information on SFS abends, see Chapter 5, "Debugging the SFS Server Machine" on page 145.

GCS Abend

For information on GCS abends, see Chapter 6, "Debugging GCS" on page 155.

TSAF Abend

For information on TSAF abends, see *VM/SP Connectivity Planning, Administration, and Operation* andChapter 7, "Debugging TSAF" on page 227 .

AVS Abend

For information on AVS abends, see *VM/SP Connectivity Planning, Administration, and Operation* and Chapter 8, "Debugging AVS" on page 235.

Virtual Machine Abend (Other than CMS)

The abnormal termination of an operating system (such as OS or DOS) 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. Because certain operating systems (such as, OS/VS1, OS/VS2, and DOS/VS) manage their virtual storage themselves, CP commands that examine or alter virtual storage locations should be used only in virtual=real storage space with systems such as, OS/VS1, OS/VS2, and DOS/VS.

The VMDUMP command dumps virtual storage to a specified virtual machine's reader spool file. The IPCS component of VM/SP may be used to process the file created by the VMDUMP command. For details, see the *VM/SP Interactive Problem Control System Guide and Reference.*

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 issue the CP IPL command. At any rate, a portion of your virtual storage is overlaid by CP's virtual IPL simulation.

If the problem can be reproduced, it may be helpful to trace the processing using the CP TRACE or CP PER commands. Also, you can set address stops, and display and alter registers, control words (such as the PSW), and data areas. The CP 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.

VM/SP may terminate or reset a virtual machine if a non-recoverable machine check occurs in that virtual machine. Hardware errors usually cause this type of virtual machine termination. The following message:

```
DMKMCH616I MACHINE CHECK; USER userid TERMINATED cpuid
```

appears on the processor console.

If the message:

```
DMKMCT621I  MACHINE CHECK; AFFINITY SET OFF
```

appears, then a machine check has occurred on the attached processor, and the attached processor is no longer being used. The virtual machine is placed into console function mode and can be made to continue processing on the main processor by the entry of a BEGIN command.

Channel checks no longer cause the virtual machine to be reset as they did in early releases of VM/370. If the problem appears to be associated with attempts to recover from a channel check, see the channel model-dependent functions described in the *VM/SP Planning Guide and Reference.*

Unexpected Results

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

Unexpected Results in CP

If an operating system executes properly on a real machine but does not execute properly with CP, a problem exists. Also, if a program executes properly under control of a particular operating system on a real machine but does not execute 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. Refer to the *VM/SP Planning Guide and Reference* 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)
- Libraries.

If the problem still is not found, look for an I/O problem. Try to reproduce the problem, while tracing all Channel Command Words (CCWs), SIOs, and interrupts with the CP TRACE or CP PER commands. Compare the real and virtual CCWs from the trace. A discrepancy in the CCWs may indicate that one of the CP restrictions was violated, or that an error occurred in CP.

Unexpected Results in a Virtual Machine

When a program executes correctly under control of a particular operating system on a real machine but has unexpected results executing under control of the same operating system with VM/SP, a problem exists. Usually you will find that something was changed. Check that the job stream, the operating system, and the system libraries are the same.

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 execute any of the utilities included in the operating system they are using to examine and rearrange files. Refer to 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) service program to print or move the data stored on direct access devices. The DDR program can be invoked by the CMS DDR command in a virtual machine controlled by CMS.

CMS users should refer to the *VM System Facilities for Programming* for instructions on using the DDR command.

Loop

The real cause of a loop is usually an instruction that sets or branches on the condition code incorrectly. The existence of a loop can usually be recognized by the ceasing of productive processing and a continual returning of the PSW instruction address to the same address. If I/O operations are involved, and the loop is a very large one, it may be extremely difficult to define, and may even include nested loops. It is probably most difficult to determine looping when caused by a wild branch.

The problem in loop analysis is finding either the instruction that should open the loop or the instruction that passed 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.

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 CP instruction currently executing in the processor. In an attached processor (AP) or multiprocessor (MP) system, trace both processors.
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, such as:
   - PSW
   - General purpose registers
   - Control registers
   - Storage locations X'00' through X'100'

   can be used to determine the condition that caused the loop.

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. That means that signalling attention does not cause an interrupt.

Enter the CP console function mode.

1. Use the CP TRACE or CP PER commands to trace the entire loop. Display general purpose and extended control registers using the CP DISPLAY command.
2. Take a dump using the CP DUMP or CP VMDUMP command. The IPCS component of VM/SP may be used to process the file created by the VMDUMP command. For details, see the *VM/SP Interactive Problem Control System Guide and Reference*.
3. 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 or CP PER commands to trace the entire loop. Display the PSW and the general purpose registers.
2. If your virtual machine has the Extended Control (EC) mode and the EC option, also display the control registers.
3. Use the CP DUMP or CP VMDUMP command to dump your virtual storage. The IPCS component of VM/SP may be used to process the file created by the VMDUMP command. For details, see the *VM/SP Interactive Problem Control System Guide and Reference*.
4. Consult the source code to search for the faulty instructions, examining previously executed modules if necessary. Begin by scanning for instructions that set the condition code or branch on it.
5. If the manner of loop entry is still undetermined, assume that a wild branch has occurred and begin a search for its origin.

Wait

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, you could also periodically issue the command:

```
#cp indicate user
```

to display the execution characteristics of the program in terms of resources used. Compare the following resources:

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

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

CP Disabled Wait

A disabled wait state usually results from a hardware malfunction. During the Initial Program Load (IPL) process, normally correctable hardware errors may cause a wait state because the operating system error recovery procedures are not accessible at this point. These conditions are recorded in the current PSW.

CP may be in an enabled wait state with channel 0 disabled when it is trying to acquire more free storage. Examine EC register 2 to see whether or not the multiplexer channel is disabled. A severe machine check could also cause a CP disabled wait state.

The following three types of severe machine checks can cause CP to terminate or cause a CP disabled wait state:

- Unrecoverable machine check in CP
- Machine check that cannot be diagnosed
- Timing facilities damage.

A machine check error cannot be diagnosed if either the machine check old PSW or the machine check interrupt code is invalid. These severe machine checks cause CP to terminate.

If a severe machine check or channel check caused a CP disabled wait state, one of the following messages appears:

```
DMKCCH603W CHANNEL ERROR
DMKMCH612W MACHINE CHECK; TIMING FACILITIES DAMAGE
DMKMCT612W MACHINE CHECK; TIMING FACILITIES DAMAGE
```

If an unrecoverable machine check occurs in CP, a message:

```
DMKMCH610W MACHINE CHECK; SUPERVISOR DAMAGE cpuid
```

-- or --

```
DMKMCT610W MACHINE CHECK; SUPERVISOR DAMAGE cpuid
```

appears on the processor console. CP is terminated and enters wait state 001 or wait state 013.

If the machine check handler cannot diagnose a certain machine check, the integrity of the system is questionable. A message:

```
DMKMCH611W MACHINE CHECK; SYSTEM INTEGRITY LOST cpuid
```

-- or --

```
DMKMCT611W MACHINE CHECK; SYSTEM INTEGRITY LOST cpuid
```

appears on the processor console. CP is terminated and enters wait state 001 or wait state 013.

Hardware errors are probably the cause of these severe machine checks. The system operator should run the CPEREP program and save the output for the installation hardware maintenance personnel.

If the generated system cannot run on the real machine because of insufficient storage, CP enters the disabled wait state with code X'00D' in the PSW. The insufficient storage condition occurs if:

- The generated system is larger than the real machine size

  -- or --

- A hardware malfunction occurs which reduces the available amount of real storage to less than that required by the generated system.

The message:

```
DMKCPJ952I nnnnnnK system storage
```

appears on the processor console. CP continues, but you should check to make sure the system is operating normally.

If CP cannot continue because consecutive hardware errors are occurring on one or more VM/SP paging devices, a message:

```
DMKPAG415E  CONTINUOUS PAGING ERRORS FROM DASD rdev
```

appears on the processor console and CP enters the disabled wait state with code X'00F' in the PSW.

If more than one paging device is available, disable the device on which the hardware errors are occurring and IPL the system again. If CP is encountering hardware errors on its only paging device, move the paging volume to another physical device and IPL again.

Note: This error condition may occur if the CP paging volume was not properly formatted.

The following procedure should be followed by the processor operator to record the needed information:

1. Using the alter/display mode of the processor console, display the real PSW and CSW. Also, display the general purpose registers and the control registers.
2. Force a CP system restart to get a system abend dump.
3. IPL the system.

Examine this information and try to find what caused the wait. If you cannot find the cause, try to reconstruct the situation that existed just before the wait state was entered.

CP Enabled Wait

If you determine that CP is in an enabled wait state, but that no I/O interrupts are occurring, there may be an error in the CP routine 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 VMBLOK for each user and the real device, channel, and control unit blocks. If each user is waiting because of a request for storage and no more storage is available, there is an error in CP. There may be looping in a routine that requests storage. Refer to "Reading CP Abend Dumps" on page 78 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:

```
DMKDSP450W   CP entered; disabled wait PSW psw
DMKDSP452W   CP entered; external interrupt loop
DMKPRG453W   CP ENTERED; PROGRAM INTERRUPT LOOP
```

and enters the console function mode. Use the CP DISPLAY command to display and obtain the following information on the terminal:

- PSW
- CSW

- General purpose registers
- Control registers.

Then use the CP DUMP or VMDUMP command to take a dump. The IPCS component of VM/SP may be used to process the file created by the VMDUMP command. For details, see the *VM/SP Interactive Problem Control System Guide and Reference*.

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 via the CP TRACE or CP PER command.

If CMS is running in the virtual machine, the CMS debugging facilities may also be used to display information or trace the processing.

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:

```
DMKDSP450W   CP entered; disabled wait PSW psw
```

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

Summary of VM/SP Debugging Commands

Table 3 summarizes the VM/SP commands that are useful for interactively debugging a problem that currently exists. The commands are classified by the function they perform. For the proper syntax of the command and a complete list of operands, refer to the appropriate command reference listed in the chart below. For the component (CP, CMS, IPCS) of each command in this list, refer to the corresponding book title.

Table 3 (Page 1 of 7). Summary of VM/SP Debugging Commands

Function	Command	Task	Refer to
Stop execution	ADSTOP PER	Stop execution at a specified location	*VM/SP CP General User Command Reference*
Resume execution	BEGIN	Use the BEGIN command to: • Resume execution where program was interrupted • Continue execution at a specific location	*VM/SP CP General User Command Reference*
Dump data	DUMP VMDUMP	Dump the contents of specific storage locations	*VM/SP CP General User Command Reference*

Table 3 (Page 2 of 7). Summary of VM/SP Debugging Commands

Function	Command	Task	Refer to
Show virtual data	DISPLAY	Use the DISPLAY command to display: • Contents of storage locations in hexadecimal. • Contents of storage locations (in hexadecimal and EBCDIC) • Storage key of specific storage locations in hexadecimal • General purpose registers • Floating point registers • Control registers • Contents of current virtual PSW in hexadecimal format • Contents of CAW • Contents of CSW	*VM/SP CP General User Command Reference*
Display real CP data	DCP	Use the DCP command to display: • Contents of processor storage locations (in hexadecimal) • Contents of processor storage locations (in hexadecimal and EBCDIC) • Contents of storage locations in IPL processor (in hexadecimal) • Contents of storage locations in non-IPL processor (in hexadecimal) • Contents of storage locations in IPL processor (in hexadecimal and EBCDIC) • Contents of storage locations in non-IPL processor (in hexadecimal and EBCDIC)	*VM/SP CP System Command Reference*
Display addresses of CP control blocks	LOCATE	Use the LOCATE command to display a: • Specified user • Virtual device • Real device	*VM/SP CP General User Command Reference*
Store virtual data	STORE	Use the STORE command to store: • Specified information into consecutive storage locations without alignment • Specified words of information into consecutive fullword storage locations • Specified words of information into consecutive general purpose registers • Specified words of information into consecutive floating- point registers • Specified words of data into consecutive control registers • Information into PSW • Information in CSW • Information in CAW	*VM/SP CP General User Command Reference*

Table 3 (Page 3 of 7). Summary of VM/SP Debugging Commands

Function	Command	Task	Refer to
Store real CP data	STCP	Use the STCP command to store: • Specified words of information into consecutive processor storage locations • Specified words of information into consecutive IPL processor storage locations • Specified words of information into consecutive non-IPL processor storage locations • Specified information into consecutive storage locations without alignment • Specified information into consecutive storage locations without alignment (in IPL processor) • Specified information into consecutive storage locations without alignment (in non-IPL processor).	*VM/SP CP System Command Reference*
Trace execution	TRACE	Use TRACE to trace: • Trace all instructions, interrupts, and branches • SVC instructions and interrupts • I/O interrupts • Program interrupts • External interrupts • Privileged instructions • All user I/O operations • Virtual and real CCWs • All user interrupts and successful branches • Instructions • End tracing activity.	*VM/SP CP General User Command Reference*
	PER	Use the PER command to trace: • Trace SVC instructions and interrupts • All user I/O operations • Successful branches • Instructions • Specific privileged instructions • Instructions that alter storage • Instructions that alter general purpose registers • Instructions that alter specific bits at specific storage locations • End tracing activity.	*VM/SP CP General User Command Reference* and *VM/SP CP System Command Reference*

Table 3 (Page 4 of 7). Summary of VM/SP Debugging Commands

Function	Command	Task	Refer to
	SVCTRACE	Use the SVCTRACE command to: • Trace SVC interrupts • End tracing activity.	*VM/SP CMS Command Reference*
Trace real machine events	MONITOR	Use the MONITOR command to: • Trace events in real machine • Stop tracing events in the real machine.	*VM/SP CP System Command Reference*
	CPTRAP	Use the CPTRAP command to: • Enable a virtual machine to enter data in CPTRAP file • Specify selectivity in collecting CPTRAP data • Define a CPTRAP debugging environment • Specify the user ID to receive CPTRAP output • Start and stop tracing • Display tracing status • Query tracing status.	*VM/SP CP System Command Reference*
Generate APAR	APAR	Use the APAR command to automatically generate a hard-copy APAR form to submit to IBM.	*VM/SP Interactive Problem Control System Guide and Reference*
Process dump and create a problem report	IPCSDUMP	Use the IPCSDUMP command to move a dump file from the reader to a CMS file. Depending on the type of dump, IPCS may try to associate a map with the file. IPCSDUMP also systematically collects information from the user to include in the dump's problem report.	*VM/SP Interactive Problem Control System Guide and Reference*

Table 3 (Page 5 of 7). Summary of VM/SP Debugging Commands

Function	Command	Task	Refer to
Format and/or print dump files or CPTRAP files	IPCSPRT	Use the IPCSPRT command to format and/or print dump files or CPTRAP files. IPCSPRT has a subcommand environment for processing CPTRAP files. The IPCSPRT subcommands are used to specify: • Types of trace entries to be printed • Time range of trace entries to be printed, • Format of trace entries to be printed. IPCSPRT has the following subcommands: • END • FORMAT • HELP • HEX • HX • PROCESS • QUIT • SELECT • TIMESPAN.	*VM/SP Interactive Problem Control System Guide and Reference*

Table 3 (Page 6 of 7). Summary of VM/SP Debugging Commands

Function	Command	Task	Refer to
Examine dump and CPTRAP files interactively	IPCSSCAN	Use the IPCSSCAN command to interactively view dumps and CPTRAP files. IPCSSCAN has a subcommand environment that has the following subcommands: REUSE, ?, + or -, &name, AREGS, ARIOBLOK, BOTTOM, C, CHAIN, CMS, CMSPOINT, CORTABLE, DISPLAY, DOSPOINT, DOWN, DUMPID, END, FDISPLAY, FORMAT, G, GDISPLAY, HELP, HEX, HX, IDENTIFY, IPCSMAP, IUCV, LOCATE, LUNAME, MAPA, MAPN, MREGS, MRIOBLOK, OSPOINT, PRINT, QUIT, REGS, RIOBLOK, SCROLL, SELECT, SYMPTOM, TACTIVE, TIME, TLOADL, TOP, TRACE, TSAB, UP, USERMAP, VIOBLOK, VMBLOK, VMLOADL.	*VM/SP Interactive Problem Control System Guide and Reference*
Process nucleus load maps	MAP	Use the MAP command to process nucleus load maps for use by IPCSDUMP.	*VM/SP Interactive Problem Control System Guide and Reference*
Update problem status	PRB	Use the PRB command to update the STATUS, FUNCTN, SEV, or DUP/APAR/PTF fields in a symptom summary record associated with a given problem number.	*VM/SP Interactive Problem Control System Guide and Reference*
Create or add to problem report	PROB	Use the PROB command to enter a problem report without using IPCSDUMP (no dump files exist) or to append information to an existing problem report. PROB, through a prompting technique, systematically collects information about the problem.	*VM/SP Interactive Problem Control System Guide and Reference*

Table 3 (Page 7 of 7). Summary of VM/SP Debugging Commands

Function	Command	Task	Refer to
List status of problems	STAT	Use the STAT command to list the current status, as found in the symptom summary file, for a given problem, a subset of problems, or all problems. Requests for the status of all problems produces a file named STATALL LOCAL that you can print. All other requests are displayed on the terminal for immediate viewing.	*VM/SP Interactive Problem Control System Guide and Reference*
List, dump, or print set of CPTRAP files for a specific problem number	TRAPFILE	Use the TRAPFILE command to list, dump, or print set of CPTRAP files for a specific problem number. TRAPFILE can either be invoked by the APAR command or by the user.	*VM/SP Interactive Problem Control System Guide and Reference*

Chapter 2. Debugging the Virtual Machine

The Control Program (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, STORE or DISPLAY) 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 execute as expected. However, when the operating system in the virtual machine itself manipulates virtual storage (for example, as in the case of OS/VS1, OS/VS2, or DOS/VS) these CP commands should not be used.

This chapter presents an overview of the VM/SP commands used for debugging. It supplements the preceding section which discussed debugging procedures and techniques. Instructions for using the commands discussed in this section are in the following:

- *VM/SP CP General User Command Reference*
- *VM/SP CP System Command Reference*
- *VM/SP Interactive Problem Control System Guide and Reference.*

The following categories of commands are discussed:

- Commands that display or dump virtual machine data
- Commands that set and query system features, conditions, and events
- Commands that trace events in virtual machines
- Commands that alter the contents of storage.

Commands that Display or Dump Virtual Machine Data

Commands that display or dump virtual machine data are: DUMP, VMDUMP, DEBUG, DISPLAY, DCP, and DMCP. See the *VM/SP CP General User Command Reference* and the *VM/SP CP System Command Reference* for more information on these commands.

DUMP

The DUMP command spools the following information to your virtual printer:

- Virtual Program Status Word (PSW)
- General purpose registers
- Floating-point registers
- Control registers (if extended control mode processing is in effect)
- Storage keys
- Virtual storage locations.

For more information on control registers see Appendix B, "Control Registers" on page 249.

When a program you execute under CMS abnormally terminates, 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.

Issue the command:

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

Then use IPCS to format and view the dump (see the *VM/SP Interactive Problem Control System Guide and Reference*).

You can selectively dump portions of your virtual storage, your entire virtual storage area, or portions of real storage. For example, when you're debugging, to dump the virtual storage space for a specified address range that may contain your program, you would enter:

```
cp dump t20000-20810
```

The second value depends upon the size of your program. Prefacing the location you want dumped with a "T," gives you an EBCDIC translation of the dump.

The DUMP command allows you to request EBCDIC translation with the hexadecimal dump.

VMDUMP

The VMDUMP command dumps virtual storage to a specified reader spool file. VMDUMP provides the same dump information that the DUMP command provides but in a different format. For example, if a byte of storage contains X'00', DUMP records it in printable format, X'F0F0'; VMDUMP records it as it appears in storage, X'00'. The IPCS component of VM/SP may be used to process the file created by the VMDUMP command (but cannot process the output from the DUMP command). For details, see the *VM/SP Interactive Problem Control System Guide and Reference*. For a description of the format and contents of the VMDUMP records, see "VMDUMP Records: Format and Content" on page 89.

DISPLAY

The DISPLAY command displays at your terminal the following kinds of control information:

- Virtual storage locations
- Storage keys
- General purpose registers
- Floating-point registers
- Control registers
- PSW
- Channel Address Word (CAW)
- Channel Status Word (CSW).

When you use the display command, you can request an EBCDIC translation of the display by prefacing the location you want displayed with a "T."

```
cp display t20000.10
```

This command requests a display of X'10' (16) bytes beginning at location X'20000'. The display is formatted with an address (20000) followed by four fullwords to a line, a storage key if you are on a page boundary, and the EBCDIC translation at the right. This is similar to what you would see in a dump.

You can also use the DISPLAY command to examine the general purpose registers, floating-point registers, and control registers. For example, the commands:

```
cp display g
cp display g1
cp display g2-5
cp display y
cp display x7
```

result in displays of all the GPRs, of GPR1, of a range of GPRs 2 through 5, of all the floating point registers, and of control register 7.

The DISPLAY command also displays the PSW, CAW, and CSW:

```
cp display psw
cp display caw
cp display csw
```

DCP and DMCP

The DCP and DMCP commands of CP are privilege class C and E commands and are used to display real storage locations. The DCP command displays at your terminal the contents of real storage locations. The DMCP command spools the contents of real storage to your virtual printer. For more information on either of these commands, please refer to the *VM/SP CP System Command Reference*.

Terminal Output

With the DISPLAY command, you can display virtual storage at your terminal in either of the following formats:

- Four-byte groups, aligned on fullword boundaries, hexadecimal format, with four fullwords per line
- 16-byte groups, aligned on 16 byte boundaries, hexadecimal format, with four fullwords plus EBCDIC translation per line.

For the first format, enter the DISPLAY command as:

```
display 1026-102c
```

you receive the response:

```
001024 xxxxxxxx xxxxxxxx xxxxxxxx
```

For the second format, enter the command as:

```
display t1026-102c
```

and the response is:

```
                                          (EBCDIC trans.)
001020 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx *...............*
```

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:

```
001024 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
001034 xxxxxxxx
```

Byte Alignment on Terminal Output

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 k1024-3200
```

the storage keys that are assigned to each 2K segment of the specified storage area are displayed. Contiguous 2K segment with identical storage keys are combined; for example, the response could have been:

```
001000 TO 0027FF  KEY=F0
002800 TO 003800  KEY=E0
```

To display all storage keys, enter:

```
display k
```

If your virtual machine is in extended control mode (ECMODE ON), you can interrogate any of the control registers:

```
display x1 4 a
```

and receive the response:

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

However, the same command entered while your virtual machine has ECMODE OFF results in the response:

```
ECR 0 = xxxxxxxx
ECR 0 = xxxxxxxx
ECR 0 = xxxxxxxx
```

As each operand in the command line is processed, VM/SP determines that ECMODE is OFF and replaces any reference to a control register with ECR 0, the only control register available in Basic Control (BC) mode.

Printer Output

With the DUMP command you can dump the contents of all available registers, the PSW and the storage keys, along with any specified area of virtual storage, to the virtual machine's spooled printer. The printer format for storage locations is 8 fullwords per line plus the EBCDIC translation on the right.

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 the beginning location and the number of bytes to be dumped; both values are entered in hexadecimal:

```
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
```

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

```
close printer
```

and the dump data spool file is placed on an appropriate system printer queue.

You can use the VMDUMP command that dumps storage for virtual machines. VMDUMP provides IPCS with header information to identify the owner of the dump; it also maintains dump information, writes the dump to a class V reader spool file, and IPCSDUMP formats the dump.

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.

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

The SYSTEM and SET commands set system-controlled functions and events; the QUERY command allows you to 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. It can also be used to clear storage.

All functions of the SET command are described in detail in the *VM/SP CP General User Command 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. You may also want to use the RUN operand.

The SET MSG function determines whether you receive messages sent by other users via 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 via the SMSG command -- when the flag is on, messages are accepted.

The SET WNG function determines whether you receive warning messages from any class A or B user.

The SET EMSG function controls error message handling. The EMSG operand gives you the ability to specify that you want message code, message text, or both to be displayed at your terminal. You can also specify that no messages be displayed (except in the case where you have spooled your console output). With EMSG on, you receive the error message numbers and the modules issuing the messages, which may be helpful to you if you are experiencing problems.

The SET IMSG command controls whether certain informational responses issued by some CP commands are displayed at the terminal or not. 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 is 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 logon, the MSG, IMSG, and WNG operands of the SET command are set ON; the SMSG operand is set OFF; the EMSG operand is set to TEXT. The RUN operand is set OFF. To verify these settings, use the QUERY SET command.

Commands that Trace Events in Virtual Machines

This section discusses the ADSTOP, TRACE, and PER commands. The TRACE command traces virtual machine events. The PER command selectively traces the execution of the instructions that cause specific events. The SVCTRACE command provides additional information about SVCs that the TRACE command does not provide. See "Using the SVCTRACE command" on page 132 for more information.

Stopping Virtual Machine Execution at a Specific Address

The ADSTOP command stops the execution of a virtual machine at a specific address; BEGIN causes the virtual machine to resume execution. See the *VM/SP CP General User Command Reference* for more information concerning the ADSTOP and BEGIN commands.

To stop execution of your virtual machine at a given address in virtual storage, use the ADSTOP command and specify the hexadecimal address of a virtual instruction. The command:

```
#cp adstop 3000
```

stops the virtual machine when the instruction at hexadecimal location 3000 is the next instruction to be executed. When the machine stops running, you receive the message:

```
ADSTOP AT 3000
```

and your terminal is placed in CP console function mode. At this point, you can enter other CP debugging commands to display and alter storage or to trace certain instructions. When you want to resume running your virtual machine, enter:

```
begin
```

Unlike the hardware address stop, ADSTOP is turned off when:

- Requested address is reached
- Next ADSTOP command is issued
- IPL or a system reset is performed
- ADSTOP OFF command is issued.

While ADSTOP is on, the SVC portion of virtual machine assist is not executed. When ADSTOP is turned off, SVCs are again handled by virtual machine assist.

The address stop should be set after the program is loaded but before it executes. When the specified location is reached during program execution, execution halts and the CP command environment is entered. You may then enter other CP commands to examine and alter the status of the program.

Set an address stop at a location where you suspect the error in the program. You can then display the registers, control words, and data areas to check the program at that point in its execution. This procedure helps you locate program errors. You may be able to alter the contents of storage in such a way that the program will execute correctly. You can then correct the error you have detected and, if necessary, compile and execute the program again.

To successfully set an address stop, the instruction must be in first-level storage of the virtual machine at the time the ADSTOP command is issued.

Using the CP TRACE Command

You can trace the following kinds of activity in a program using the CP TRACE command:

- Instructions (privileged and PSW)
- Branches
- Interrupts
  - Program
  - External
  - I/O
  - SVC
- I/O and channel activity.

See the *VM/SP CP General User Command Reference* for more information concerning the TRACE command.

When the TRACE command executes, it traces all your virtual machine's activity; when your program issues a supervisor call, or calls any CMS routine, the TRACE continues. Trace output for the CP TRACE command is always produced *before* the instruction executes.

Tracing resumes when these two conditions are met:

- CP gains control, such as for a real I/O interruption, and
- Virtual machine encounters one of the specified activities to be traced, except for successful branching.

Whenever you are recording trace output at your terminal, the virtual machine stops execution and enters the CP console read environment after each output line. This is the default mode of operation when, for example, you enter:

```
trace all
```

or

```
trace svc program branch
```

If you only want to record the trace and not stop after each output line, add the RUN operand as the last entry on the command line.

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

```
trace program branch off
```

and tracing is now confined to SVCs only.

To trace all activity with the output directed to the virtual printer, enter:

```
trace all printer
```

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

```
trace end
close printer
```

If your virtual machine configuration contains only one printer, trace output is intermixed with application output. You should define another virtual printer with an address lower than the previously defined printer. Application output is still directed to the original printer; however, trace output is always directed to the printer with the lowest address.

While trace is running, portions of virtual machine assist are disabled. When the trace is complete, they are enabled.

You can make most efficient use of the TRACE command by starting the trace at a specific instruction location. You should set an address stop for the location. For example, if you are going to execute a program and you want to trace all of the branches made, you would enter the following sequence of commands to begin executing the program and start the trace: (Remember, commands are represented in lower case, responses are in uppercase.)

```
load progress
cp adstop 20004
start
DMSLIO740I Execution begins ...
ADSTOP AT 20004
cp trace branch
cp begin
```

Now, whenever your program executes a branch instruction, you receive information at the terminal that might look like this:

```
02001E BALR  05E6    ==> 020092
```

This line indicates that the instruction at address X'2001E' resulted in a branch to the address X'020092'. When this information is displayed, your virtual machine is placed in the CP environment, and you must use the BEGIN command to continue execution:

```
cp begin
```

When you locate the branch that caused the problem in your program, you should terminate tracing activity by entering:

```
cp trace end
```

and then you can use CP commands to continue debugging.

Controlling a CP Trace

There are several things you can do to control the amount of information you receive when you are using the TRACE command, and how it is received. For example, if you do not want program execution to halt every time a trace output message is issued, you can use the RUN option:

```
cp trace svc run
```

Then, you can halt execution by pressing the Attention key when the interruption you are waiting for occurs. You should use this option if you do not want to halt execution at all, but merely want to watch what is happening in your program.

Similarly, if you do not require your trace output immediately, you can specify that it be directed to the printer, so that your terminal does not receive any information at all:

```
cp 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 at a virtual address lower than that of your printer, usually at 00E. For example:

```
cp define printer 006
```

Then, trace output will be in a separate spool file. CMS printed output always goes to the printer at address 00E.

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

```
cp close e
```

or

```
cp close 006
```

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

```
cp trace all both
```

Suspending Tracing

If you are debugging a program that does a lot of I/O, or that issues many SVCs, and you are tracing instructions or branches, you might not wish to have tracing in effect when the supervisor or I/O routine has control. When you notice that addresses being traced are not in your program, you can enter:

```
cp trace end
```

and then set an address stop at the location in your program that receives control when the supervisor or I/O routine has completed:

```
cp adstop 20688
begin
```

Then, when this address is encountered, you can re-enter the CP TRACE command.

What To Do When Your Program Loops

If your program seems to be in a loop, you should first verify that it is looping, and then interrupt its execution and either:

- Halt it entirely and return to the CMS environment, or
- Resume its execution at an address outside of the loop.

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

You can verify a loop by checking the PSW frequently. If the last word repeatedly contains the same address, it is a fairly good indication that your program is in a loop. You can check the PSW by using the Attention key (except with 3270 type terminals) to enter the CP environment. You are notified by the message:

```
CP
```

that your virtual machine is in the CP environment. You can then use the CP command DISPLAY to examine the PSW:

```
cp display psw
```

and then enter the command BEGIN to resume program execution:

```
cp begin
```

If you are checking for a loop, you might enter both commands on the same line using the logical line end:

```
cp display psw#begin
```

When you have determined that your program is in a loop, you can halt execution using the CMS Immediate command HX. To enter this command, you must press the Attention key once (except with 3270 type terminals) to interrupt program execution, then enter:

```
hx
```

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

```
cp begin 20cd0
```

Or, you could use the CP command STORE to change the instruction address in the PSW before entering the BEGIN command. For example,

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

Debugging with CP After a Program Check

When a program that is executing under CMS abends because of a program check, the DEBUG routine is in control and saves your program's registers, so that if you want to begin debugging, you may use the DEBUG command. See the *VM/SP CMS Command Reference* for the information you can receive from the DEBUG command.

You can prevent DEBUG from gaining control when a program interruption occurs by setting the wait bit in the program new PSW:

```
cp trace prog norun
```

You should do this before you begin executing your program. Then, if a program check occurs during execution, when CP tries to load the program new PSW, the wait bit forces CP into a disabled wait state and you receive the message:

```
start
EXECUTION BEGINS...
***024602 PROG 0001 ==> 1E3D18
```

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, at location X'28'. Use the DISPLAY command to examine its contents:

```
cp display 28.8
```

The program new PSW, or the PSW you see if you enter the command DISPLAY PSW, contains the address of the DEBUG routine.

```
1 INSTRUCT RANGE 020000-0204FF TERMINAL NORUN
```

If in addition to instructions, you wish to trace instructions that alter registers, enter:

```
per g range 20000.500
```

To see which events you are monitoring, enter:

```
query per
```

You will see the following:

```
1 INSTRUCT RANGE 020000-0204FF TERMINAL NORUN
2 G RANGE 020000-0204FF TERMINAL NORUN
```

To change *just* the instruction trace element to PRINTER, you can enter:

```
per instruct range 020000-0204ff printer run
```

To see which events you are currently monitoring, enter:

```
query per
```

You will see the following:

```
1 G RANGE 020000-0204FF TERMINAL RUN
2 INSTRUCT RANGE 020000-0204FF PRINTER RUN
```

If you continue program execution by entering BEGIN you will receive information at your terminal that might look like this:

```
020004 BALR 05C0   000000 CC=0 G12=40020006
```

This line indicates a BALR instruction at address X'020004' changed register 12 to X'40020006'.

As with CP TRACE, you can specify the printer and/or run for any event. However, CP PER has additional options that can be used with all events:

- The RANGE or FROM option can be used to set up multiple instruction address ranges. This can increase the selectivity with which instruction execution is monitored.
- The PASS option allows you to suppress a specific number of events between displays.
- The CMD option can be used if you want to execute CP command(s) whenever a given event occurs.
- The STEP option can be used to permit a specified number of events to be displayed before the CP command environment is entered.

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 at location X'28'. For example, if the instruction address in your program is X'566' enter:

```
cp store psw 0 20566
cp 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 execution can resume.

Using the CP PER Command

The CP PER command can be used to trace all:

- Instructions
- Successful branches
- Register alterations
- Instructions executed in your virtual machine that alter storage.

See the *VM/SP CP General User Command Reference* and the *VM/SP CP System Command Reference* for more information concerning the PER command.

The CP PER command has many options that allow you selectivity in choosing which events are to be monitored. Trace output for the CP PER command is always produced *after* the instruction executes. The RANGE keyword of the CP PER command can be used to set multiple address stops. Note also that address stops set using the PER command remain in effect until you turn off the trace element set up by the PER command. There is no need for the program to already be in storage before setting address stops with the CP PER command.

Setting up multiple address stops with PER is accomplished by using RANGE as an option to the INSTRUCT keyword. The instruction-addr-range, in this case, is a single value corresponding to the address of the instruction where program execution is to be halted.

For example,

```
per instruct range 20000
```

causes program execution to halt after the instruction at location X'20000' executes.

```
per instruct range 20000 range 20400
```

causes a program to halt after an instruction at either location X'20000' or X'20400' executes.

Note: Although output is produced only after the instruction at X'20000' or X'20400' executes, the hardware causes a PER interrupt for every instruction executed in the range X'20000' to X'20400'. This may degrade the performance of the virtual machine.

The CP QUERY command with the PER option can be used to determine what events are currently being traced. For example:

```
query per
```

may result in:

Selectivity

PER options can be used to increase selectivity. Using PER, 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:

```
per instruct data 18
```

When the NORUN option is in effect, program execution halts after each monitored event. When using the RUN option, program execution continues after each event. PER also has an execution rate between NORUN and RUN. This option is called STEP. STEP specifies the number of events that should be displayed before program execution halts and the CP command environment is entered. For example, to halt program execution after 5 instructions in the range X'20000' to X'204FF' have been executed, enter:

```
per instruct range 20000.500 step 5
```

When your program has been loaded and started, you will receive information at your terminal that might look like this:

```
==>020000 STM   90ECD00C 00DFAC CC=0
  020004 BALR  05C0     000000 CC=0
  020006 ST    50D0C342 020348 CC=0
  02000A LA    41D0C33E 020344 CC=0
  020012 OC    D607C3A2C3AA 0203A8 0204B0 CC=1
```

and then program execution would halt, and the CP command environment would be entered.

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

```
per instruct pass 100
```

Terminating PER

To end PER tracing with the current set of events, enter:

```
per end current
```

To end just the branch trace elements in the current set of events, enter:

```
per end branch
```

It is not always desirable to end all the trace elements of a particular event type. When the current set of events being traced is displayed using the QUERY PER command, each trace element is preceded by a number. This number can be used to selectively end PER tracing.

To see which events you are currently monitoring, enter:

```
query per
```

You may see the following:

```
1 INSTRUCT PRINTER RUN
2 G5 DATA 00000023 TERMINAL NORUN
3 G7 TERMINAL NORUN
4 STORE INTO 020628-020630 TERMINAL NORUN
```

If you no longer want to monitor instructions that alter register 7 but want to continue monitoring register 5, enter:

```
per end 3
```

To terminate all PER tracing, enter:

```
per end all
```

This ends the current and all saved sets of events being traced.

Suspending PER

The PER SAVE subcommand can be used to save the current set of events being traced. This saved set is saved only while the user is logged on.

To save the current set under the name TRACE1, enter:

```
per save trace1
```

The current set is still active. To suspend the PER tracing, enter:

```
per end current
```

PER tracing is no longer active. The set of events is saved under the name TRACE1. You can now execute normally, or create another current set using PER commands. This new set (or sets) of events can also be saved.

To resume PER tracing with the original set, enter:

```
per get trace1
```

A copy of set TRACE1 is still saved under the name of TRACE1. Changes made to TRACE1 while it is the current set have no effect on this saved copy.

To end the saved set TRACE1, enter:

```
per end trace1
```

If you save a set under the same name as an existing set, the current set replaces the old set unless you specify the append option.

Additional Program Debugging Using PER

TRACE can be used to trace program interrupts. However, the trace information is displayed after the interrupt has occurred and cannot always be used to determine the cause of the problem. PER, used in conjunction with TRACE, can greatly reduce the difficulty of finding the cause of the problem. If the problem is an operation exception, it may have been caused by a bad branch instruction.

The Branch Traceback Table

The first step is to trace program interrupts using TRACE:

```
trace prog
```

Run the failing program until the program interrupt occurs. When the program interrupt occurs, the address of the instruction causing the interrupt is the address that precedes the address of the instruction that is being displayed. For example:

```
start
EXECUTION BEGINS...
***024602 PROG 0001 ==> 1E3D18
```

Next end TRACE and allow the program to finish. Reload the failing program and trace successful branches to the address of the bad instruction. For example:

```
per branch 24600
```

Note: The branch might be to an address before X'24600'. The branch might have encountered a valid operation code. Therefore, it is sometimes necessary to use a larger branch into address. For example:

```
per branch 245f0-24600
```

When the branch to the bad instruction occurs, the branch instruction as well as the previous 5 successful branches are displayed. For example:

```
start
EXECUTION BEGINS...
==>020012 BR    07F1     024600 CC=0
TRACEBACK TABLE:
:1D1320 BR    07F3     1D125A
:1D1268 BR    07FE     1D1322
:1D1356 BNZ   4770E07C 1D139E
:1D13A2 BZ    4780E090 1D13B2
:1DFE98 BR    07FF     020000
```

Note: If control is transferred to the bad address by a LPSW or an interrupt (for example an SVC) PER BRANCH does not trace this event. Therefore, it is a good idea to issue a TRACE PROG before starting the program. Then, if the program interrupt occurs before any PER output is produced, the PER TABLE command can be used to display the branch traceback table containing the last 6 successful branches. The last entry in the table is the last successful branch instruction executed before the program interrupt. While this is not necessarily the instruction causing the problem, hopefully it is near the failing instruction. It is now possible to restart the program using PER to trace the execution of instructions in the range beginning with this branch instruction, and ending at the program interrupt address.

The PER COUNT Subcommand

Another method of finding the failing instruction is to use the PER COUNT sub-command with TRACE. This method, as well as the use of the PER TABLE command, is well suited for problems other than just operation exceptions. If the program is abending with any sort of program exception, then load the failing program, and issue the CP command:

```
trace prog
```

followed by:

```
per instruct range 20000.500
```

(assuming the program is X'500' bytes in length) and then:

```
per count
```

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

```
query per
```

You may see the following:

```
1 INSTRUCT RANGE 020000-0204FF TERMINAL NORUN
  PER COUNT 2159
```

This means that 2159 instructions were executed before the instruction that caused the program interrupt. It is now possible to trace as many instructions leading up to the program interrupt as desired. To trace the last 15 instructions before the program interrupt, reload the failing program, and issue the following PER command:

```
per pass 2144
```

the response is:

```
PER COUNT 2159
PER COUNT ENDED
```

This command has two effects. First, it turns off the PER COUNT option, and second it applies the PASS option to the current set of events. The current set now contains:

```
1 INSTRUCT RANGE 020000-0204FF TERMINAL NORUN PASS 2144
```

Next start the failing program. The first 2144 instructions executed in the range X'20000' through X'204FF' are not displayed. The 2145th instruction is displayed. When the instruction is displayed, issue:

```
per pass
```

This command resets the PASS option to the default (display every instruction). The current set now contains:

```
1 INSTRUCT RANGE 020000-0204FF TERMINAL NORUN
```

It is now possible to trace the last 15 instructions, and to use the DISPLAY command to display storage and register contents.

PER COUNT can also be used in conjunction with more specific trace elements to produce the desired results. For example, if a problem occurs as a result of the execution of an SVC 202 and the failing program issues many SVC 202s before failing, it may not be productive to use TRACE.

An alternative is to use PER to set up a set of events that traces only SVC 202s (operation code X'0ACA') and to use PER COUNT to count the occurrences. First, load the failing program and then issue:

```
per instruct 0aca range 20000.500
per count
```

and start the program. When the failure occurs, issue a QUERY PER to check the count.

```
query per
```

You may see the following:

```
1 INSTRUCT 0ACA RANGE 020000-0204FF TERMINAL NORUN
  PER COUNT 623
```

The program can then be traced after using the PER PASS option as above to get close to the problem.

The PER Command Option

The PER CMD option can be used to execute any CP command (except SLEEP) whenever a particular event occurs. For example:

```
per instruct range 20000.500 run
per store 204f0-204ff range 20000.500 run cmd display 204f0-204ff
```

traces the execution 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, the CMD option can be used to cause execution of a program to continue at a specific address whenever a particular event occurs. For example:

```
per instruct range 20000.500 printer
per branch 0 run cmd begin 24f28
```

causes program execution to continue at location X'24F28' whenever a branch to location 0 occurs. The execution continues after the instruction is displayed. If, when program execution is resumed at location X'24F28', a subsequent branch to zero occurs, execution again begins at location X'24F28'. This can result in a loop. The CMD option can also be used to prevent this. For example, if LINEDIT is on, and the escape character is set to " and the line end character is #, then:

```
per instruct 20000.500 printer
per branch 0 run cmd per end branch"#begin 24f28
```

turns off the branch trace element and causes program execution to continue at location X'24F28' after the instruction is displayed. The current set would then be:

```
1 INSTRUCT RANGE 020000-0204FF PRINTER RUN
```

The commands associated with each trace element are executed whenever the event described by the trace element occurs. The commands are executed in the order in which they appear in the set of events. Therefore, if the current set is:

```
1 INSTRUCT TERMINAL RUN CMD cmd1#cmd2
2 BRANCH TERMINAL RUN CMD cmd3
3 G TERMINAL RUN CMD cmd4#cmd5
```

and an instruction is executed that alters a register, but does not cause a successful branch, then the CP commands cmd1, cmd2, cmd4, and cmd5 are executed (in that order).

Note: If a CP command is entered while commands are being executed by PER, the output from the commands may be interleaved.

Once the command option has been specified for a particular trace element, the command option remains in effect until the trace element is turned off. However, the command can be changed. For example, if the current set contains:

```
1 BRANCH TERMINAL RUN CMD DISPLAY G15
2 G TERMINAL RUN CMD DISPLAY PSW
```

the command associated with the branch trace element can be changed to DISPLAY G14-15 by issuing:

```
per branch terminal run cmd display g14-15
```

or both commands can be changed to DISPLAY G14-15 by issuing:

```
per cmd display g14-15
```

Storage Alteration

PER can be used to trace the alteration of storage in the user's virtual machine. If PER STORE is specified, then whenever an instruction places a value into storage, that event is traced. 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:

```
per 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 the execution of the instructions.

It is also possible to monitor only alterations of storage when the storage value actually changes. For example:

```
per mask into 20100 data ffffffffffffffff
```

monitors instructions that actually change one or more bytes in the doubleword field starting at location X'20100'. PER MASK can also be used to monitor changes to specific bits in storage. For example:

```
per mask into 20100 data 8040
```

monitors instructions that change the status of the first bit in the byte at location X'20100' or the second bit in the byte at location X'20101'.

GUESTR and GUESTV

PER traces virtual machine activity in both second- and third-level storage. Using the GUESTR and GUESTV options, it is possible to choose which level activity will be traced. For example, when debugging VM/SP within a virtual machine it is sometimes helpful to limit trace output to either just second- or just third-level activity. CP itself runs with Dynamic Address Translation (DAT) off. When CP dispatches a virtual machine, the DAT bit in the PSW is on. Therefore, if CP is IPLed into a virtual machine, then that CP is executing in second-level storage. When CMS is IPLed on top of the second-level CP, then that CMS is executing in third-level storage. If the command:

```
per i range 20000-21000
```

is issued, then both second- and third-level activity is traced (that is, both CP and CMS). To only trace the CP activity (second level) in the range X'20000' through X'21000', enter:

```
per i range 20000-21000 guestr
```

To only trace the CMS activity (third level) in the range X'20000' through X'21000', enter:

```
per i range 20000-21000 guestv
```

It is also possible to selectively trace specific activity in both second- and third-level storage. For example, to trace successful branches and storage alterations of location X'1024' in second-level storage and to trace branches to location X'20000', and alterations to register 7 in third-level storage, enter:

```
per branch store into 1024 guestr
per branch into 20000 g7 guestv
```

Commands that Alter the Contents of Storage

You can use the STORE and STCP commands to alter the contents of virtual machine storage and real storage.

ZAP and ZAPTEXT commands are used to alter TEXT libraries or TEXT decks before the code is loaded and executed.

STORE and STCP are described in the following paragraphs. See *VM/SP Installation Guide* for information on ZAP and ZAPTEXT.

Altering the Contents of Virtual Machine Storage (STORE command)

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

- Virtual machine storage locations
- General purpose registers
- Floating-point registers
- Control registers (if available)
- Program Status Word.

The STORE STATUS command can save certain information contained in low storage.

When debugging, you may find it advantageous to alter storage, registers, or the PSW and then continue execution. This is a good procedure for testing a proposed change. Also, you can make a temporary correction and then continue to ensure that the rest of execution is trouble-free. A procedure for using the STORE STATUS command when debugging is as follows:

- Issue the STORE STATUS command before entering a routine you wish to debug.
- When execution stops (because an address stop was reached or because of failure), display the extended logout area. This area contains the status that was stored before entering the routine.
- Issue STORE STATUS again and display the extended logout area again. You now have the status information before and after the failure. This information should help you solve the problem.

Altering Virtual Storage

You can alter the contents of your virtual storage, GPRs, floating-point registers, control registers (if available), and the PSW with the STORE command.

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:

```
store 1024 46a2
```

or

```
store l1024 46a2
```

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

```
store 1024 46 a2
```

on the other hand, implies storing 2 fullwords and results in the storing of X'00000046000000A2' in locations X'1024' through X'102B'.

If the starting location is not a multiple of a fullword, it is automatically rounded down to the next lower fullword boundary. Each fullword operand can be from one to eight hexadecimal digits in length. If less than 8 digits are specified, they are right justified in the fullword unit and padded to the left with zeros.

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

```
store s1026 d1d6c5
```

stores X'D1D6C5' in locations X'1026', X'1027', and 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 terminates 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,

```
store g4 123456
```

loads GPR 4 with X'00123456'.

```
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'.

You can use the STATUS operand of the STORE command to simulate the hardware store status facility. Selected virtual machine data is stored in permanently assigned areas in low storage. Your virtual machine must be in extended control mode for the command:

```
store status
```

to be accepted. To place your virtual machine in extended control mode, issue the command:

```
set ecmode on
```

Be aware that this command resets your virtual machine and you must reload (IPL) your operating system.

The data stored by the STORE STATUS command is summarized in the following table:

Virtual Dec.	Address Hex	No. of Bytes	Data
216	D8	8	Processor Timer
224	E0	8	Clock Comparator
256	100	8	Current PSW
352	160	32	Floating-Point Registers (0,2,4, and 6)
384	180	64	General Purpose Registers (0-15)
448	1C0	64	Control Registers (0-15)

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.

Altering the Contents of Real Storage (STCP command)

Use the STCP command to alter the contents of real storage. The STCP command can alter the old and new PSWs, but the user has to know where in storage these are located.

Chapter 3. Debugging CP

Commands to Collect and Analyze System Information

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

- MONITOR
- INDICATE
- QUERY SRM
- LOCATE.

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

See *VM/SP Administration*, *VM/SP CP General User Command Reference*, and *VM/SP CP System Command Reference* for more information on the MONITOR, INDICATE, and QUERY SRM commands.

Locating CP Control Blocks in Storage

Use the class C or E 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. The control blocks and their functions are described in the *VM/SP CP Data Areas and Control Blocks.*

Once you know the location of the control blocks, you can examine the block you want to look at. When you want to examine specific control blocks, use the DCP command to display or the DMCP command to print the control blocks. A discussion of the most important fields of the VMBLOK, VCHBLOK VCUBLOK, VDEVBLOK, RCHBLOK, RCUBLOK, and RDEVBLOK are included in "Virtual and Real Control Block Status" on page 83.

Debugging CP in a Virtual Machine

Many CP problems can be isolated without stand-alone machine testing. It is possible to debug CP by running it 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 is used to generate 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 *VM Running Guest Operating Systems* for directions on how to set up the virtual machine.

CP Internal Trace Table

CP has an internal trace table that records events that occur in the real machine. The events that are traced are:

- External interrupts
- SVC interrupts
- Program interrupts
- Machine check interrupts
- I/O interrupts
- Free storage requests
- Release of free storage
- Entry into scheduler
- Queue drop
- Run user requests
- Start I/O
- Unstack I/O interrupts
- Storing a virtual CSW
- Test I/O
- Halt Device
- Unstack IOBLOK or TRQBLOK
- Network Control Program (NCP) Basic Transmission Unit (BTU)
- Spinning on a lock (AP or MP environment)
- SIGP issued
- Clear Channel instruction
- IUCV communications
- SNA Console Communication Services (CCS)
- MSSF DIAGNOSE code X'80'
- Start I/O fast release
- Simulated I/O interruptions
- Clear I/O
- Time stamp
- Test channel.

An installation may optionally specify the size of the CP internal trace table. To do so, use the SYSCOR macro in module DMKSYS. Information on using this macro instruction is in the *VM/SP Planning Guide and Reference*.

If an installation does not specify the CP internal trace table size or the size specified is smaller than the default size, VM/SP CP assigns the default size.

Type of Event	Module	IO Code (hex)[1]	Format of Trace Table Entry 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
External Interrupt	DMKEXT	01	X'01' | 5X'00' | Interrupt Code | External Old PSW
SVC Interrupt	DMKSVC	02	X'02' | GPR 14 or GPR 15[2] | Instruction Length Code[5] | SVC Interrupt Code[5] | SVC Old PSW
Program Interrupt	DMKPMA[7] DMKPRG	03	X'03' | First 3 bytes of VMPSW | Instruction Length Code[5] | Interrupt Code | Program Old PSW
Machine Check Interrupt	DMKMCH	04	X'04' | VMBLOK Address | First 4 bytes of 8-byte Interrupt Code | Machine Check Old PSW
I/O Interrupt	DMKIOT	05	X'05' | Measurement Byte | Device Address | I/O Old PSW + 4 | CSW
Obtain Storage (FREE)	DMKFRE	06	X'06' | VMBLOK Address | GPR 0 at entry | GPR 1 at exit | GPR 14
Return Storage (FRET)	DMKFRE DMKFRT	07	X'07' | VMBLOK Address | GPR 0 at entry | GPR 1 at entry | GPR 14
Enter Scheduler	DMKSCH	08	X'08' | VMBLOK Address | VMR-STAT | VMD-STAT | VMO-STAT | VMQ-STAT | VMQ-LEVEL | VMT-LEVEL | VMIOINT | VMPEND | GPR 14
Queue Drop	DMKSCH	09	X'09' | VMBLOK Address | VMEPRIOR | DMK-SCHAL | VMU-PRIOR | VMQPRIOR | VMUHS
Run User	DMKDSP	0A	X'0A' | 3X'00' | RUNUSER value from PSA | RUNPSW value from PSA
Start I/O	DMKACS DMKCNS DMKCPM DMKCPU[7] DMKIOS DMKIOT[7] DMKMNT	08	X'08' | CC | Device Address | IOBLOK Address | CAW | For CC = 1, CSW + 4. Otherwise, this field is 4X'00'.
Unstack I/O Interrupt	DMKDSP	0C	X'0C' | X'00' | Virtual Device Address | VMBLOK Address | Virtual CSW
Virtual CSW Store	DMKVSJ	0D	X'0D' | Instruction OP Code | Virtual Device Address | VMBLOK Address | Virtual CSW
Test I/O	DMKACS DMKCPM DMKCPU[7] DMKIOS DMKIOT[7] DMKMNT	0E	X'0E' | CC | Device Address | IOBLOK Address | CAW | For CC = 1, CSW + 4. Otherwise, this field is 4X'00'.
Halt Device	DMKCNS DMKIOS DMKMNT DMKVSJ	0F	X'0F' | CC | Device Address | IOBLOK Address | CAW | For CC = 1, CSW + 4. Otherwise, this field is 4X'00'.
Unstack IOBLOK or TRQBLOK	DMKDSP	10	X'10' | VMBLOK Address | VMR-STAT | VMD-STAT | VMO-STAT | VMQ-STAT | IOBLOK or TRQBLOK Address | Interrupt Return Address
NCU BTU[3]	DMKRNH	11	X'11' | X'00' | CONSRID | CONDEST | CONRTAG | CON-SYSR | CON-EXTR | CONTCMD | CON-FUNC | CON-DFLG | CONDCNT
Spinning on Lock	DMKLOK	12	X'12' | VMBLOK Address | Lockword Address | Return Address | Lockword Contents
SIGP Issued	DMKEXT	13	X'13' | Return Address | X'00' | CC | Real Processor Address | Function Code[4] | Order Code | Status of CC = 1
Clear Channel Issued	DMKVSJ	14	X'14' | CC | Device Address | VMBLOK Address | Virtual CSW
IUCV Communication	DMKIUA	15	X'15' | Function Code[4] | Path Id or Unused | IUCVBLOK Address | Use varies by Function Code (See *VM System Facilities for Programming (SC24-5288)* for details on IUCV trace table format.) | IUCV Instruction Address
SNA CCS	DMKVCV DMKVCX	16	X'16' | Transaction Type | See *VM System Facilities for Programming (SC24-5288)* for details on SNA CCS entries.
MSSF DIAGNOSE X'80'	DMKMHC	17	X'17' | CC | 2X'00' | HCBLOK Address | MSSF Command Word | MSFBLOK Address
Start I/O Fast Release	DMKIOS	18	X'18' | CC | Device Address | IOBLOK Address | CAW | For CC =1, CSW + 4. Otherwise, this field is 4X'00'.
Simulated I/O Interrupt	DMKIOT	19	X'19' | X'00' | Device Address | DMKDID or DMKACS Address in GPR 12 at entry | CSW
Run User Through Fast Path[6]	DMKDSP	1A	X'1A' | 3X'00' | RUNUSER value from PSA | RUNPSW value from PSA
Clear I/O	DMKIOS	1B	X'1B' | CC | Device Address | IOBLOK Address | CAW | For CC = 1, CSW + 4. Otherwise, this field is 4X'00'.
Resetting VM Pages[6]	DMKPTS	1C	X'1C' | VMBLOK Address | VMR-STAT | VMD-STAT | VMO-STAT | VMQ-STAT | VMS-WSTAT | VMS-WPFL1 | VMS-WPFL2 | X'00' | GPR 14
Logical Swap-In[6]	DMKSWAP1	1D	X'1D' | VMBLOK Address | VMR-STAT | VMD-STAT | VMO-STAT | VMQ-STAT | VMQ-LEVEL | VMT-LEVEL | VMS-WPL1 | VMS-WSTAT | VMRSWPGS | SC8NWS
Obtain Prime Storage[6]	DMKFRE	1E	X'1E' | VMBLOK Address | GPR 0 at entry | GPR 1 at exit | GPR 14
Return Prime Storage[6]	DMKFRE DMKFRT	1F	X'1F' | VMBLOK Address | GPR 0 at entry | GPR 1 at exit | GPR 14
CPTRAP Time Stamp	DMKTRT	20	X'20' | Reserved | Time-of-day Stamp
Test Channel	DMKIOS	21	X'21' | CC | Device Address | IOBLOK Address | CAW | Not Used

Notes:
1. If the installation is running in AP or MP mode, the identification code will be ORed with a X'40' if the activity occurred on the non-IPL processor. If the installation is running ECPS, the identification code is ORed with an X'80' if the activity occurred in microcode.
2. If the interrupt code (bytes 6 and 7) is X'000C', the contents of GPR 14 are displayed. For all other interrupt codes, the contents of GPR 15 are displayed.
3. Bytes 2 through 15 of a code 11 trace record represent a Basic Transmission Unit, sent or received by a 37XX. If CONSYSR/CONEXTR are zero, the BTU was transmitted to the 37XX. If they are non-zero, the BTU was received. If CONTCMD equals X'7700', this is an unsolicited BTU response.
4. For the SIGP instruction, trace table entry 13 byte 8 contains the function code for emergency signal and external call order codes.
5. For VM/SP HPO without ECPS, byte 4 contains the Interrupt Code and bytes 5, 6, and 7 contain GPR 13. For SVC 8 and SVC 20, GPR 13 on exit. For SVC 12 and SVC 16, GPR 13 on entry.
6. These trace table entries are for VM/SP HPO only.
7. These modules are for VM/SP HPO only.

Figure 5. VM CP Trace Table

For each 256K bytes (or part thereof) of real storage available at IPL time, one page (4096 bytes) is allocated to the CP internal trace table. Each entry in the CP internal trace table is 16 bytes long. There are CP internal trace table entries for each type of event recorded. The first byte of each CP internal trace table entry, the identification code, identifies the type of event being recorded. Figure 5 on page 75 describes the format of each type of CP internal trace table entry. See the *VM CP Trace Table (Poster)* for a 20 inch by 26 inch poster of the CP internal trace table. Also see the *VM/SP Problem Determination Summary* for a reference card that may be easily carried that shows the CP internal trace table. The entry shown in Figure 5 on page 75 for IUCV communications illustrates the general format of an IUCV entry. See *VM System Facilities for Programming* for the formats of the CP internal trace table entries for each IUCV function, and for a description of each field in the CP internal trace table entry.

In addition, some CP internal trace table entries are generated by Extended Control Program Support:VM/370 (ECPS:VM/370). The first bit of these entries is set to 1 to indicate the entry was generated by the hardware assist. For example, a CP internal trace table entry of type X'86' (FREE) is the same as an entry of type X'06'. The only difference is that the first entry was generated by the hardware assist.

The CP internal trace table is allocated by DMKSTA which is called by the main initialization routine, DMKCPI. The first event traced, TRACSTRT, is placed in the lowest CP internal trace table address. Each subsequent event is recorded in the next available CP internal trace table entry. Once the CP internal trace table is full, events are recorded at the lowest address (overlaying the data previously recorded there). Tracing continues with each new entry replacing an entry from a previous cycle.

Use the CP internal trace table to determine the events that preceded a CP system failure. An abend dump contains the CP internal trace table and the pointers to it. The address of the start of the CP internal trace table, TRACSTRT, is at location X'0C'. The address of the byte following the end of the CP internal trace table, TRACEND, is at location X'10'. The address of the next available CP internal trace table entry, TRACCURR, is at location X'14'. Subtract 16 bytes (X'10') from the address stored at X'14' (TRACCURR) to obtain the CP internal trace table entry for the last event completed.

The CP internal trace table is initialized during IPL. If you do not wish to record events in the CP internal trace table, issue the MONITOR STOP CPTRACE command to suppress recording. The pages allocated to the CP internal trace table are not released and recording can be restarted at any time by issuing the MONITOR START CPTRACE command. If the VM/SP system should abnormally terminate and automatically restart, the tracing of events on the real machine will be active. After a VM/SP IPL (manual or automatic), CP internal tracing is always active.

Abend Dumps

There are three kinds of abnormal termination dumps possible when using CP. The first kind occurs if the problem program cannot continue, it terminates and, in some cases, tries to issue a dump.

The second kind occurs if the operating system for your virtual machine cannot continue, it terminates and, in some cases, tries to issue a dump. In the virtual machine environment, the problem program dump always goes to the virtual printer. Depending on installation operating procedures, the virtual machine operating system dump may also go to the virtual printer. A CLOSE must be issued to the virtual printer to have either dump print on the real printer.

The third type of dump occurs when the CP system cannot continue. CP abend dumps can be directed to a:

- Printer
- Tape
- DASD.

If the dump is directed to a printer, then the data is printed online and system operations as well as virtual machine operations are suspended until the dump operation finishes. This dump is unformatted.

If the dump is directed to a tape, the dumped data must fit on one reel of tape. VM/SP does not support multiple tape volumes for dumps. The data on the tape is in print-line format and can be processed by user-created programs or CMS commands. See "How to Print a CP Abend Dump from Tape" on page 78 for an example of how CMS can do this.

If the dump is directed to DASD, it is done by spooling the data to the virtual card reader of a specific user ID. The user ID is either assigned during system generation to a specific virtual machine user, or defaults to the printer. The dump spool file can be manipulated by the user just like any other spool file, except that it can be interpreted correctly only by the IPCSDUMP command (for more details see the *VM/SP Interactive Problem Control System Guide and Reference*). By issuing the class B QUERY DUMP command you can determine where the dump is being directed.

The extent of the CP abend dump can be specified to dump:

- CP storage only
- CP and all real storage

to the selected device.

Use the CP SET DUMP command to specify the output device and extent of CP abend dumps. Refer to the *VM/SP CP System Command Reference* for the format of the class B SET DUMP command.

How to Print a CP Abend Dump from Tape

If the CP dump unit has been specified as a tape drive, and one or more dumps have been placed on the tape, use the following procedure to print the dumps:

1. Log on to the system with any user ID that has the capability of running CMS. No other special privilege classes or options are required.
2. Attach a tape drive to the virtual machine as address 181 from an authorized user.
3. Mount the tape that has the CP abend dumps.
4. IPL the CMS system.
5. Issue the following CMS commands, filling in the variable names:

   FILEDEF *ddname1* PRINTER (RECFM FM LRECL 132)
   FILEDEF *ddname2* TAP1 (DEN *den* RECFM U LRECL 132)
   MOVE *ddname2 ddname1*
   CP CLOSE PRT

   Note: Refer to the FILEDEF command description in *VM/SP CMS Command Reference* if you need help filling in the variable used in the example above.

Step 5 can be repeated for as many dumps as are on the tape. Note that the CP dump routines write two tape marks at the end of each file. Therefore, to process the next dump, the TAPE FSF command must be issued to position the tape for reading the next dump file.

Reading CP Abend Dumps

Two types of printed dumps occur when CP abnormally ends, depending upon the options specified in the CP SET DUMP command.

First, when the dump is directed to a direct access device, Interactive Problem Control System (IPCS) must be used to format and print the dump. For IPCS use, see the *VM/SP Interactive Problem Control System Guide and Reference*. IPCS commands format and print:

- Control blocks
- General purpose registers
- Floating-point registers
- Control registers
- TOD (Time-of-Day) Clock
- Processor Timer
- Storage
- If in AP or MP mode, formats and prints both PSAs' storage.

Storage is printed in hexadecimal notation, eight 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.

If the CP SET DUMP command directed the dump to a tape or a printer, the printed format of the printed dump will not contain formatted control blocks. If the system was an attached processor or multiprocessor, all of the registers, etc., are

printed for the abending processor. Also, each PSA is printed before printing main storage.

Second, when CP can no longer continue and abnormally terminates, 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. See *VM/SP Administration* for information that will help you understand the major functions of CP. The following discussion on reading CP dumps includes many references to CP control blocks and control block fields. Refer to *VM/SP CP Data Areas and Control Blocks* or for a description of the CP control blocks. Figure 6 on page 84 shows the CP control block relationships. Also, you will need the current load map for CP to be able to identify the modules from their locations. The load map is created at initial CP generation time. See the *VM/SP Installation Guide* to get information on saving and printing the CP load map.

Reason for the Abend

Determine the immediate reason for the abend. You need to examine several fields in the Prefix Storage Area (PSA), to find the reason for the abend. In a uniprocessor system, the PSA is in page 0. In an Attached Processor (AP) or Multiprocessor (MP) system, each processor has its own PSA in addition to the absolute PSA in page 0.

1. Examine the program old PSW and program interrupt code to find whether or not a program check occurred in CP. The program old PSW (PROPSW) is located at X'28' and the program interrupt code (INTPR) is at X'8E'. If a program check has occurred in supervisor mode, use the CP system load map to identify the module. If you cannot find the module using the load map, refer to "Identifying and Locating a Pageable Module" on page 88. Figure 18 on page 251 describes the format of an Extended Control PSW.

2. Examine the SVC old PSW, the SVC interrupt code, and the abend code to find whether or not a CP routine issued an SVC 0. The SVC old PSW (SVCOPSW) is located at X'20', the SVC interrupt code (INTSVC) is at X'8A', and the abend code (CPABEND) is at X'374'.

   The abend code (CPABEND) is a fullword. The first three bytes identify the module that issued the SVC 0 and the fourth byte is a binary field whose value indicates the reason for issuing an SVC 0.

   Use the CP system load map to identify the module issuing the SVC 0. If you cannot find the module using the CP system load map, refer to "Identifying and Locating a Pageable Module" on page 88. Figure 18 on page 251 describes the format of an Extended Control PSW.

3. Examine the restart old PSW (RSRTOPSW) at X'08'. If an abnormal termination occurs because the operator caused a system restart, the old PSW at location X'08' points to the instruction that was executing when CP recognized the abnormal termination. Figure 18 on page 251 describes the format of an Extended Control PSW.

4. For a machine check, examine the machine check old PSW and the logout area. The machine check old PSW (MCOPSW) is found at X'30' and the fixed logout area (FXDLOG) is at X'100'. Also examine the machine check interrupt code (INTMC) at X'E8'.

Collect Information

Examine several other fields in the PSA to analyze the status of the system. As you progress in reading the dump, you may return to the PSA to pick up pointers to specific areas (such as pointers to the real control blocks) or to examine other status fields. For specific fields within the PSA control block, refer to *VM/SP CP Data Areas and Control Blocks.*

The following areas of the PSA may contain useful debugging information:

1. The CP running status is stored in CPSTAT at location X'348'. The value of this field indicates the running status of CP since the last entry to the dispatcher.
2. Current User

   The PSW that was most recently loaded by the dispatcher is saved in RUNPSW at location X'330', and the address of the dispatched VMBLOK is saved in RUNUSER at location X'338'. Also, examine the contents of control registers 0 and 1 as they were when the last PSW was dispatched. See RUNCR0 (X'340') and RUNCR1 (X'344') for the control registers.

Also, examine the CP internal trace table to determine the events that preceded the abnormal termination. Start with the last event recorded in the trace table and read backward through the trace table entries. The last event recorded is the last event that was completed.

The TRACSTRT field (location X'0C') contains the address of the start of the trace table. The TRACEND field (location X'10') contains the address of the byte following the end of the trace table. The address of the next available trace table entry is found in the TRACCURR field (location X'14'). To find the last recorded trace table entry, subtract X'10' from the value at location X'14'. The result is the address of the last recorded entry. Figure 5 on page 75 describes the format of each type of trace table entry.

Note: If the system was in AP or MP mode, the trace table pointers are in absolute page zero.

Register Use

To trace control blocks and modules, it is necessary to know the CP register-use conventions.

The 16 General Purpose Registers (GPRs) have many uses that vary depending upon the operation. The following table shows the use of some of the general purpose registers:

Register	Contents
GPR 1	The virtual address to be translated.
GPR 2	The real address or parameters.
GPR 6,7,8	The virtual or real channel, control unit, and device control blocks.
GPR 10	The address of the active IOBLOK.
GPR 14, 15	The external branch linkage.

The following general purpose registers usually contain the same information:

Register	Contents
GPR 11	The address of the active VMBLOK.
GPR 12	The base register for the module executing.
GPR 13	The address of the current save area if the module was called via an SVC.

Use these registers along with the CP control blocks and the data in the prefix storage area to determine the error that caused the CP abend.

Save Area Conventions

The save areas that may be helpful in debugging CP are: BALRSAVE, FREESAVE, FREEWORK, and DUMPSAVE, all in PSA; and SAVEAREA, which is not in PSA. If a module was called by an SVC, examine the SAVEAREA storage area. SAVEAREA is its own control block (documented in *VM/SP CP Data Areas and Control Blocks*) and the address of the SAVEAREA is found in general purpose register 13. If a module was called by a branch and link, the general purpose registers are saved in the PSA in an area called BALRSAVE (X'240'). The work area and save area for DMKFRE and DMKFRT are also in the PSA; these areas are used only by the DMKFRE and DMKFRT routines. The save area (FREESAVE) for DMKFRE and DMKFRT can be found at location X'280' and the work area (FREEWORK) follows at location X'2C0'.

Save areas used by attached processor and multiprocessor support are SIGSAVE, LOKSAVE, MFASAVE, SWTHSAVE, LOCKSAV, and SVCREGS. These save areas are all in the PSA. LOCKSAV and SVCREGS are four fullwords in size; the others are 16 fullwords in size.

Use the save areas to trace backwards and find the previous module executed.

1. SAVEAREA

   An active save area contains the caller's return address in SAVERETN (displacement X'00'). The caller's base register is saved in SAVER12 (displacement X'04'), and the address of the save area for the caller is saved.

2. BALRSAVE

   All the general purpose registers are saved in BALRSAVE after branching and linking (via BALR) to another routine. Look at BALR14 for the return address saved, BALR13 for the caller's save area, and BALR12 for the caller's base register, and you can trace module control backwards.

3. FREESAVE

   All the general purpose registers are saved in FREESAVE before entries in DMKFRE or DMKFRT execute. Use this address to trace module control backwards.

Field	Contents
FREER15	The entry point (in DMKFRE or DMKFRT).
FREER14	The saved return address.
FREER13	The caller's save area (unless the caller was called via BALR).
FREER12	The caller's base register.
FREER1	Points to the block returned (for FRET entries).
FREER0	Contains the number of doublewords requested or returned.

4. DUMPSAVE

   All the general purpose registers at the time of the error are saved in DUMPSAVE (displacement X'500') before DMKDMP is called. They are saved by DMKPSA after a restart, by DMKSVC after an SVC 0, and by DMKPRG. The registers are stored in DUMPSAVE in the order GPR0 through GPR15. GPR12 usually contains the base register for the module executing at the time of the error.

5. SIGSAVE

   SIGSAVE (displacement X'540') is used as a save/work area by DMKEXT, a MP/AP-only module that handles all signaling requests. When a signal request is issued, DMKEXTSP is called. On entry, DMKEXTSP stores GPR0 through GPR6 and GPR12 through GPR15. GPR7 through GPR11 are not saved. The remainder of SIGSAVE is used as a work area. GPR14 contains the caller's return address.

6. LOKSAVE

   All the general purpose registers are stored in LOKSAVE (displacement X'580') before DMKLOK executes. DMKLOK is an MP/AP-only module that manipulates certain locks. The registers are stored in the order GPR0 through GPR15. GPR14 contains the caller's return address.

7. MFASAVE

   All the general purpose registers are stored in MFASAVE (displacement X'5C0') before DMKMCTMA executes. DMKMCTMA is the entry into DMKMCT, an MP/AP-only module, that handles malfunction alert interrupts. The registers are stored in the order GPR0 through GPR15. GPR14 and GPR15 contain the caller's return address.

8. SWTHSAVE

   All the general purpose registers are stored in SWTHSAVE (displacement X'600') by DMKSTK and DMKVMASW. DMKVMASW is an entry that is used only in MP/AP systems to switch a user's page table pointers. The registers are stored in the order GPR0 through GPR15. GPR14 contains the caller's return address. All entries to DMKSTK store registers GPR0 through GPR15 in SWTHSAVE.

9. LOCKSAV

   LOCKSAV (displacement X'640') is a four-word save area used by the LOCK macro to save GPR14, GPR15, GPR0, and GPR1 if the SAVE option of the LOCK macro is specified.

10. SVCREGS

    SVCREGS (displacement X'650') is a four-word save area used to save GPR12 through GPR15 at the time of an SVC interrupt.

Virtual and Real Control Block Status

Examine the virtual and real control blocks for more information on the status of the CP system. Figure 6 on page 84 describes the relationship of the CP control blocks; several are described in detail in the following paragraphs. For even more detail on the following control blocks, refer to *VM/SP CP Data Areas and Control Blocks*.

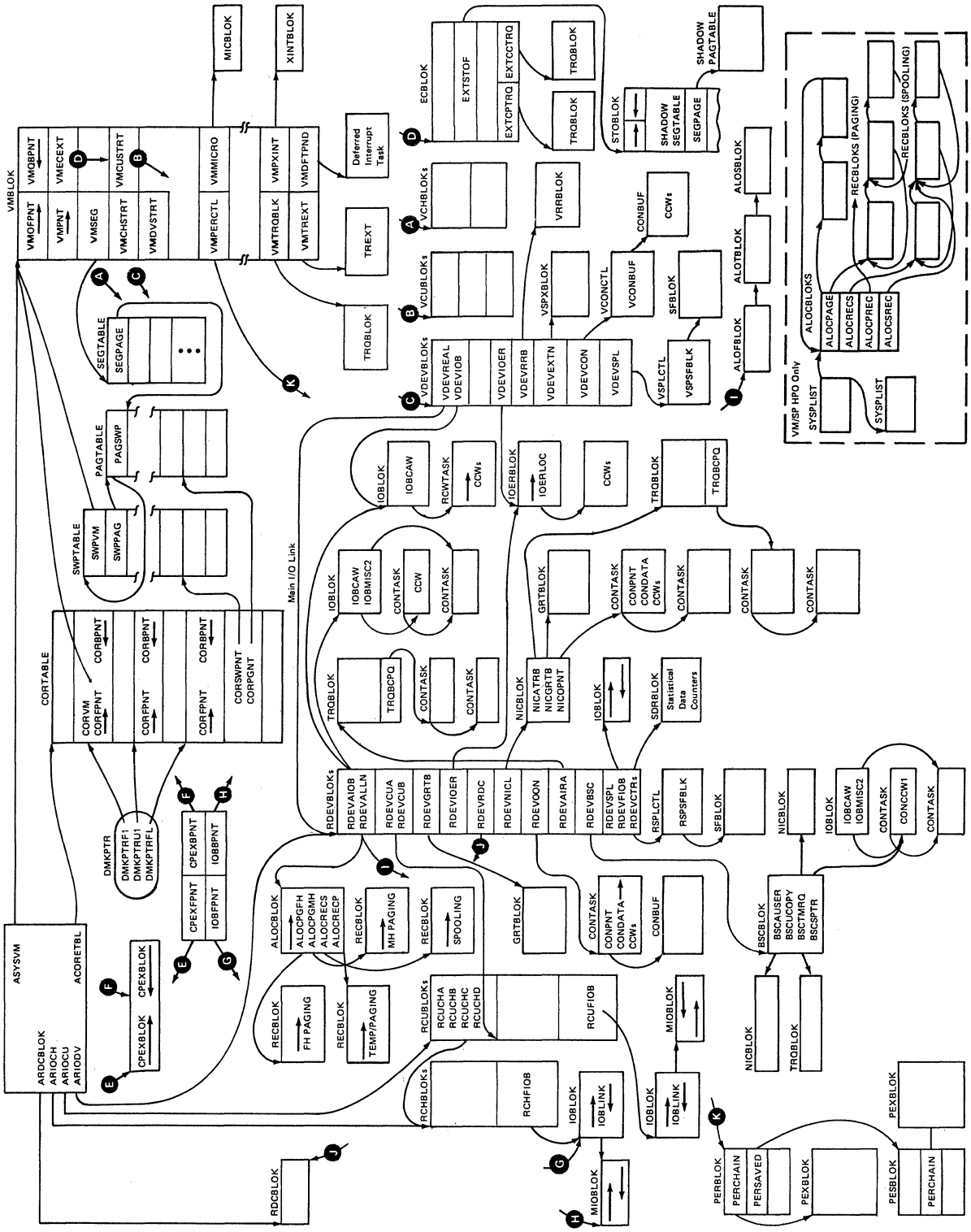


Figure 6. CP Control Block Relationships

VMBLOK

The address of the VMBLOK is in general purpose register 11.

Examine the following VMBLOK fields:

1. VMRSTAT (displacement X'58') contains the virtual machine running status.
2. VMDSTAT (displacement X'59') contains the virtual machine dispatching status.
3. VMINST (displacement X'98') saves the virtual machine privileged or tracing instruction.
4. VMPSW (displacement X'A8') saves the virtual machine PSW.
5. VMCOMND (displacement X'148') contains the name of the last CP command that executed.
6. For checking the status of I/O activity, the following fields contain pertinent information.
   a. VMIOACTV (displacement X'36') is the active channel mask. Each bit represents a channel (0 through 15). An active channel is indicated by a 1 in the corresponding bit position.
   b. VMPEND (displacement X'63') contains the interrupt pending summary flag. The value of VMPEND identifies the type of interrupt.
   c. VMFSTAT (displacement X'68') contains the virtual machine feature status.
   d. VMIOINT (displacement X'6A') contains the I/O interrupt pending flags. Each bit represents a channel (0 through 15). An interrupt pending is indicated by a 1 in the corresponding bit position.

VCHBLOK

The address of the VCHBLOK table is found in the VMCHSTRT field (displacement X'18') of the VMBLOK. General purpose register 6 contains the address of the active VCHBLOK. Examine the following fields:

1. VCHADD (displacement X'00') contains the virtual channel address.
2. VCHSTAT (displacement X'06') contains the status of the virtual channel.
3. VCHTYPE (displacement X'07') contains the virtual channel type.

VCUBLOK

The address of the VCUBLOK table is found in the VMCUSTRT field (displacement X'1C') of the VMBLOK. General purpose register 7 contains the address of the active VCUBLOK. Useful information is contained in the following fields:

1. VCUADD (displacement X'00') contains the virtual control unit address.
2. VCUSTAT (displacement X'06') contains the status of the virtual control unit.
3. VCUTYPE (displacement X'07') contains the type of the virtual control unit.

VDEVBLOK

The address of the VDEVBLOK table is found in the VMDVSTRT field (displacement X'20') of the VMBLOK. General purpose register 8 contains the address of the active VDEVBLOK. Useful information is contained in the following fields:

1. VDEVADD (displacement X'00') contains the virtual device address.
2. VDEVSTAT (displacement X'06') contains the status of the virtual device.
3. VDEVFLAG (displacement X'07') contains the device-dependent information.
4. VDEVCSW (displacement X'08') contains the virtual channel status word for the last interrupt.
5. VDEVEXTN (displacement X'10') is the pointer to the virtual spool extension block, VSPXBLOK, for output spooling devices.
6. VDEVREAL (displacement X'24') is the pointer to the real device block, RDEVBLOK.
7. VDEVSFLG (displacement X'27') describes the virtual spooling flags for spooling devices.
8. VDEVIOB (displacement X'34') is the pointer to the active IOBLOK.
9. VDEVFLG2 (displacement X'38') describes the Reserve/Release flags and other miscellaneous conditions.
10. VDEVRRB (displacement X'3C') contains the address of the VRRBLOK for Reserve/Release minidisks.
11. VDEVCFLG (displacement X'49') describes the virtual console flags for console devices.

RCHBLOK

The address of the first RCHBLOK is found in the ARIOCH field (displacement X'3B4') of the PSA. General purpose register 6 contains the address of the active RCHBLOK. Examine the following fields:

1. RCHADD (displacement X'00') contains the real channel address.
2. RCHSTAT (displacement X'04') describes the status of the real channel.
3. RCHTYPE (displacement X'05') describes the real channel type.
4. RCHFIOB (displacement X'08') is the pointer to the first IOBLOK in the queue and RCHLIOB (displacement X'0C') is the pointer to the last IOBLOK in the queue.

RCUBLOK

The address of the first RCUBLOK is found in the ARIOCU field (displacement X'3B8') of the PSA. General purpose register 7 points to the current RCUBLOK. Examine the following fields:

1. RCUADD (displacement X'00') contains the real control unit address.
2. RCUSTAT (displacement X'04') describes the status of the control unit.
3. RCUTYPE (displacement X'05') describes the type of the real control unit.
4. RCUFIOB (displacement X'08') points to the first IOBLOK in the queue.

5. RCULIOB field (displacement X'0C') points to the last IOBLOK in the queue.
6. RCUCHA (displacement X'10') is the pointer to the Primary RCHBLOK.
7. RCUCHB (displacement X'14') is the pointer to the first alternate RCHBLOK.
8. RCUCHC (displacement X'18') is the pointer to the second alternate RCHBLOK.
9. RCUCHD (displacement X'1C') is the pointer to the third alternate RCHBLOK.

RDEVBLOK

The address of the first RDEVBLOK is found in the ARIODV field (displacement X'3BC') of the PSA. General purpose register 8 points to the current RDEVBLOK. Also, the VDEVREAL field (displacement X'24') of each VDEVBLOK contains the address of the associated RDEVBLOK. Examine the following fields of the RDEVBLOK:

1. RDEVADD (displacement X'00') contains the real device address.
2. RDEVSTAT (displacement X'04'), RDEVSTA2 (displacement X'45'), and RDEVSTA4 (displacement X'60') describe the status of the real device.
3. RDEVFLAG (displacement X'05') indicates device flags. These flags are device-dependent.
4. RDEVTYPC (displacement X'06') describes the device type class and the value of the RDEVTYPE field (displacement X'07') describes the device type. Refer to "Appendix A" in *VM/SP CP Data Areas and Control Blocks* for the list of possible device type class and device type values.
5. For spooling unit record devices, RDEVSPL (displacement X'18') is the pointer to the active RSPLCTL block.
6. For real 37xx Communications Controllers, several pointers are defined. RDEVEPDV (displacement X'1C') is the pointer to the start of the free RDEVBLOK list for EP lines. RDEVNICL (displacement X'38') is the pointer to the network control list and RDEVCKPT (displacement X'3C') is the pointer to the CKPBLOK for re-enable. Also, RDEVMAX (displacement X'2E') is the highest valid NCP resource name and RDEVNCP (displacement X'30') is the reference name of the active 37xx NCP.
7. RDEVAIOB (displacement X'24') contains the address of the active IOBLOK.
8. RDEVUSER (displacement X'28') is the pointer to the VMBLOK for a dedicated user.
9. RDEVATT (displacement X'2C') contains the attached virtual address.
10. For terminals, an additional flag is defined. RDEVTMCD (displacement X'34') describes the line code translation to be used.
11. For terminal devices, additional flags are defined. RDEVTFLG (displacement X'3A') describes the additional flags.
12. RDEVIOER (displacement X'48') contains the address of the IOERBLOK for the last CP error.

Identifying and Locating a Pageable Module

If a program check PSW or SVC PSW points to an address beyond the end of the CP resident nucleus, the failing module is a pageable module. The CP system load map identifies the end of the resident nucleus, labeled as DMKCPEND.

Go to the address indicated in the PSW. Backtrack to the beginning of *that* page frame. The first eight bytes of that page frame (the page frame containing the address pointed to by the PSW) contains the name of the first pageable module loaded into the page. If multiple modules exist within the same page frame, identify the module using the load map and failing address displacement within the page frame. In most cases, register 12 points directly to the name.

To locate a pageable module whose address is shown in the load map, use the system VMBLOK segment and page tables. For example, if the address in the load map is 55000, use the segment and page tables to locate the module at segment 5, page 5.

Debugging an AP/MP System

When you debug an AP/MP problem, the following areas provide pertinent information:

- PSA
- Trace table
- Lockwords.

PSA

A dump for a program operating in AP or MP mode contains three PSAs:

- Absolute PSA
- One PSA (address is in PREFIXA at X'660') for the IPL processor
- One PSA (address is in PREFIXB at X'664') for the other processor.

In a formatted dump, the PSA for the IPL processor is displayed first and the PSA for the other processor is displayed second. The PSA contains important information about the status of each processor, such as:

- Normal low-core IPL
- Logout
- PSW information
- Processor model, type, and features
- BALR areas
- FREE areas
- Monitor and trace data
- Linkages to virtual machines, real devices, and spool files.

See *VM/SP CP Data Areas and Control Blocks* for layout and explanation of the fields in the PSA.

Trace Table

In an AP/MP system, the trace table entries for both processors are intermixed. However, you can identify which processor made a particular entry by looking at the trace code in the first byte of the trace table entry. If bit 1 of the trace code contains a zero, the entry was made by the IPLed processor; but if bit 1 of the trace code contains a 1, the entry was made by the other processor. Processor identification information is implemented for an AP/MP system at system initialization when the system assigns each processor a trace identifier. The system

assigns the IPLed processor a trace identifier of X'00' and the non-IPLed processor a trace identifier of X'40'. The identifier is ORed with the trace code when an entry is made in the trace table thus providing an easy way of determining which processor made a particular entry.

The following trace table entries appear in an AP/MP environment:

X'12' indicates that the processor is spinning on a lock
X'13' indicates that a processor issued a signal processor (SIGP) instruction
X'01' may reflect multiprocessing-related external interruption codes (also appears in a uniprocessor environment)

When you are debugging an AP/MP system, you must relate the entries made by one processor to the entries made by the other processor in the same time period. For example, a signal processor (code X'13') entry by one processor should be followed closely by an external interruption (code X'01') for the other processor. See Figure 5 on page 75 for the layout of the CP internal trace table. The CP internal trace table pointers:

- Trace table start address
- Trace table end address
- Next available trace entry address

are in absolute page zero.

Lockwords

You can look in the DMKLOK module to find the status of the various VM/SP locks except the VMBLOK lock and the RDEVBLOK lock. Each of the locks in DMKLOK contains the following four fullwords of information:

- First fullword contains the logical processor address of the owning processor. This will be zero if the lock is not held.
- Second fullword contains the value in the lock owner's register 12.
- Third fullword contain the total amount of time spent spinning on this lock.
- Fourth fullword contains the total number of spins.

The VMBLOK lock is located in the VMBLOK at VMLOCK (X'1A8'). When the VMBLOK lock is held, VMLOCK contains the logical processor address of the owning processor.

The RDEVBLOK lock is located in the RDEVBLOK at RDEVIOBL (X'54'). When the lock is held, RDEVIOBL contains the logical processor address of the owning processor.

VMDUMP Records: Format and Content

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

CP writes the storage dump to the spool file as a series of logical records. Each spool file record and each logical dump record is 4096-bytes long. However, because each spool file record contains a header, one logical dump record does not fit into

one spool file record. For this reason, CP splits a logical dump record into two parts. CP writes one part to one spool file record and the other part to an adjacent spool file record. The size of each part varies depending upon the amount of space remaining in the spool file record that CP is currently using. Thus, each logical dump record spans two spool file records. Figure 7 on page 92 shows the format of spool file records, the format of logical dump records, and how logical dump records span spool file records.

The first spool file record contains a spool page buffer linkage block (SPLINK) header followed by a TAG area followed by dump information. All other spool file records contain only a SPLINK header followed by dump information.

A SPLINK header, which contains data needed to locate information in the associated spool file record, has the following format:

Hexadecimal Offset	Length	Content
0	4 bytes	DASD location (DCHR) of the next page buffer
4	4 bytes	DASD location (DCHR) of the previous page buffer
8	4 bytes	Binary zeroes
C	4 bytes	Number of data records in the buffer

Following the SPLINK header, the TAG area contains either binary zeroes or user supplied data. If a virtual machine program or the user has issued the TAG command, the TAG area contains the information provided via this command. Otherwise it contains binary zeroes.

The first logical dump record contains a dump file information record (DMPINREC). The second and third logical dump records each contain a dump file key storage record, DMPKYREC1 and DMPKYREC2, respectively. The dump file key storage records contain the value of the storage keys assigned to each page of virtual storage. The remaining logical dump records contain the virtual machine storage dump.

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.

For a description of the format and contents of DMPINREC, see *VM/SP CP Data Areas and Control Blocks*. For a description of DMPKYREC1 and DMPKYREC2, see DMPKYREC also in *VM/SP CP Data Areas and Control Blocks*.

The VMDUMP command dumps virtual storage that VM/SP created for the virtual machine user. VMDUMP creates a file that provides IPCS with header information to identify the owner of the dump. Once VMDUMP creates the file, IPCS may process it to debug errors, as well as to store and maintain error information about the virtual machine. For additional information, see the *VM/SP Interactive Problem Control System Guide and Reference*.

Locating Logical Dump Records

To locate a specific logical dump record, use the algorithm:

$$loc = \frac{240+16n+4096n+[((16n+224)/4080)x16]}{4096}$$

n
: is a number that identifies the dump record. For example, to locate the first dump record, assign n a value of 1; to locate the second record, assign n a value of 2, and so forth.

/
: represents integer division.

loc
: is the quotient and remainder of the algorithm.

Together these values specify a spool file record and an offset into that record where logical dump record *n* begins. The quotient specifies the spool file record, and the remainder specifies the offset into the spool file record.

The following example shows how to locate the third logical dump record:

$$loc = \frac{240+(16x3)+(4096x3)+[(((16x3)+224)/4080)x16]}{4096}$$

$$loc = \frac{12576+0}{4096}$$

quotient = 3

remainder = 288

Thus, the third dump record starts 288 bytes into the third spool file record.

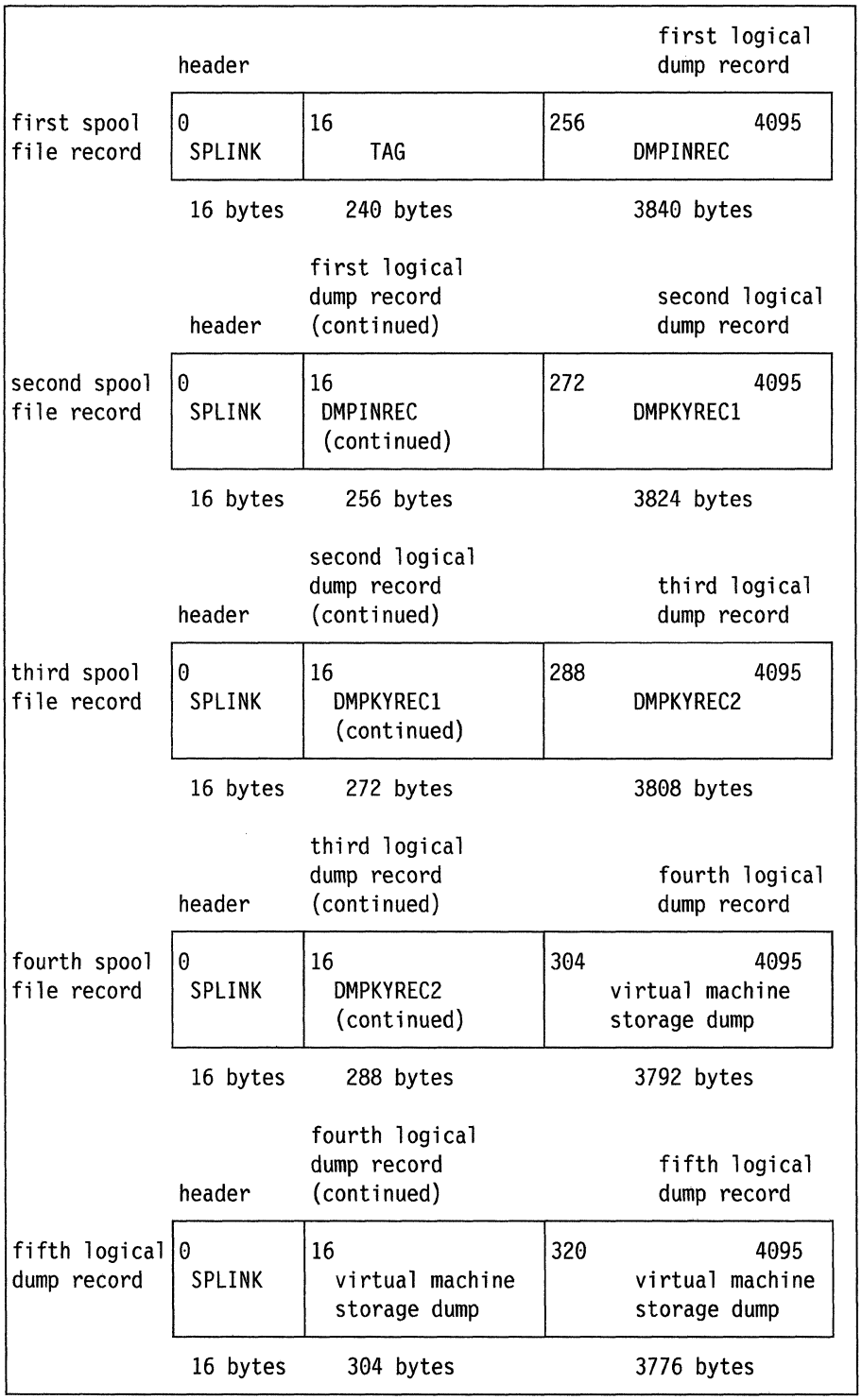


Figure 7. VMDUMP Record Format

Trapping Improper Use of CP Free Storage

Installations can install the CP FRET Trap as an aid in solving problems caused by improper use of CP free storage and to solve many storage overlay problems. The design of the CP FRET Trap allows it to produce "tracks" in storage associated with each free storage request. The trap detects the release of areas of free storage that were not assigned, previously released, or outside the boundaries of the storage given.

The trap code is conditionally assembled in the DMKFRE, DMKFRT, and DMKCPI modules based on the value of the option &FRETRAP. Found in OPTIONS COPY, &FRETRAP has a default value of 0 for normal operations without the trap.

The CP FRET Trap does the following:

- Disables CP Assist FREE, FRET, DSP1, DSP2, and UNTFR instructions.
- Expands each request for free storage by a three doubleword extension containing:
  - The status of the request. The status consists of the tag ALLO when the storage is allocated by DMKFRE or the tag FRET when the storage is released by DMKFRT.
  - The saved size (in doublewords) of the requested free storage area.
  - The address of the assigned free storage block.
  - The return address of the module requesting the storage.
  - The last three characters of the calling module's name (if it is pageable).
  - The user's VMBLOK address.
  - The rest of the extension is cleared with zeroes and remains unused until the storage is released. The content at that time is as follows:
    - The return address of the module releasing the storage.
    - The last three characters of the calling module's name (if it is pageable).

  Note: For the exact format of the extension, refer to the FREEXT control block in the *VM/SP CP Data Areas and Control Blocks.*
- Checks each request to release free storage for the expected tag. Checks the size of the free storage area to be released against the saved size in the extension area, and abends in illegal situations. The ALLO tag is replaced with the FRET tag if the trap detects no problems with the FRET request.

When the CP FRET Trap is installed, performance for systems using CP Assists is degraded due to the disabling of the DSP1, DSP2, UNTFR, FREE, and FRET instructions. Also, performance for storage constrained systems having many users is degraded due to the expansion of each free storage request to include the trap extension area. The performance degradation is not likely to be a problem while suspected free storage problems are being trapped. The overall performance of the system remains the same when the trap is not installed.

The trap has to be installed at system generation time. Refer to the *VM/SP Installation Guide* for installation instructions and the *VM/SP CP Diagnosis Reference* or for specifics on the logic.

CP FRET Trap Examples

The following two examples demonstrate how the trap may be used to solve problems caused by improper use of CP free storage.

Example 1: Destruction of the free storage pointer.

Symptom:

Module X obtains a 36 doubleword block of storage from DMKFREE. The data in the storage block is being overlaid by data that has no resemblance to the data expected to be there.

The CP FRET Trap is installed and it abends with code FRT015.

Analyzing the Available Data:

The trap found the ALLO tag at FREER1 + the value of FREER0 in bytes. It abended with code FRT015 because the saved size of the original request (in the trap extension area) did not match the size of the block to be released in FREER0. Examination of FREER12, FREER14, FREER0, and FREER1 reveals that module Y called DMKFRET to release 40 doublewords of storage at the address contained in FREER1. Further examination of the trap extension area shows that module X made the original request for the free storage and that the requested size was 36 doublewords.

Conclusion:

If the free storage pointer in FREER1 and the size in FREER0 were correct, the size would have matched the saved size in the extension area. If the free storage pointer in FREER1 were correct and the size in FREER0 incorrect, then the trap would not have found the ALLO tag at FREER1 + FREER0. The trap would have abended with code FRT013 instead of FRT015. Therefore, the free storage pointer in FREER1 was incorrect when module Y tried to release the storage block.

The free storage block of module X could have been overlaid by module Y when its free storage pointer was destroyed. Or, when the trap was not installed, the storage block could have been released by module Y, recycled by DMKFREE and reissued to module Z, causing an overlay of the storage obtained by module X.

Example 2: Release of more storage than was given.

Symptom:

Module Y obtains a 9 doubleword block of storage from DMKFREE. The data in the storage block is being overlaid by data that has no resemblance to the data expected to be there.

The CP FRET Trap is installed and it abends with code FRT013.

Analyzing the Available Data:

The trap abended with code FRT013 because it could not find the ALLO tag at FREER1 + the value of FREER0 in bytes. Examination of FREER12, FREER14, FREER0, and FREER1 shows that module X called DMKFRET to release 15 doublewords of storage at the address contained in FREER1. Examining the storage at FREER1 reveals that an ALLO tag can be found at

FREER1 + 9 doublewords, and that the saved size in the extension is 9 doublewords. The VMBLOK address in the extension matches that in FREER11. Further examination of the storage for the next ALLO tag shows that the storage block obtained by module Y overlaps the storage being released by module X by 3 doublewords.

Conclusion:

Module X tried to release more storage than was actually given. The free storage block of module Y could have been overlaid when the size for the storage block being released by module X was incorrect. Or, when the trap was not installed, the storage block could have been released by module X, recycled by DMKFREE and reissued to module Z, causing an overlay of the storage obtained by module Y.

Debugging with the CPTRAP Facility

Overview

The CPTRAP facility can be used to create a reader spool file of selected trace table entries, CP data, I/O data, or virtual machine data in the order they happen. This data is collected in 4K blocks and placed in the CPTRAP spool file (CPTRAP FILE). A X'20' time stamp record is inserted into the CPTRAP spool file and CP internal trace table each time data is copied from the CP internal trace table to the CPTRAP spool file. (The format of the X'20' time stamp record is shown in Figure 5 on page 75.) Each record may have multiple entries.

Note: CPTRAP FILE is the file name and file type assigned to the CPTRAP spool file by the CPTRAP facility.

The CPTRAP facility uses the concept of **traps**. A **trap** is used to tell CPTRAP which events in CP or a virtual machine should generate data recording in the CPTRAP file. What is recorded depends on the type of trap. This recording of data about events occurring while a program is running is referred to as **tracing**.

The facility provides the following types of traps:

- The IO type trap allows you to examine all of the I/O activity to any devices or range of devices. For any transaction, the following data is recorded with the CSW: the channel program, and, optionally, the data transferred. This trap can be used to collect data associated with the interfaces between CP and I/O devices.
- The TTABLE type trap allows you to control which CP trace entries are collected in the CP internal trace table and the CPTRAP spool file. You can also alter the trace flags in the PSA with this type trap. This trap should be used to filter trace table data to obtain just the internal trace table entries needed.
- The GT type trap allows you to record virtual machine events, including guest virtual machine activity, in the CPTRAP spool file.
- The DATA type trap allows you to examine data conditions during execution of nearly any path in CP. This trap can be used to collect data associated with communications between CP and guest operating systems and components in CP.

You may issue the CPTRAP command with the DISPLAY operand to show you the current status of the trap IDs you are working with. Also you can use the QUERY CPTRAP ALL command to obtain the same information. For more information on the QUERY CPTRAP command, see *VM/SP CP System Command Reference.*

You can use the VM/SP Interactive Problem Control System (VM/SP IPCS) to access the CPTRAP reader file and the data collected in the file. VM/SP IPCS provides an interactive, online facility for reporting and diagnosis of software failures and for managing problem information and status. It assists in reporting problems, diagnosing problems, and managing problems and related data. Refer to the *VM/SP Interactive Problem Control System Guide and Reference* for additional information on how IPCS can be used to process CPTRAP files.

Defining a Trap

You define traps by using the definition operands (ID operand) of the CPTRAP command. You can use operands associated with ID to give the trap a name (trap ID) and optionally with the SET operand to group the trap with other traps that can be enabled, disabled, or dropped together. In addition, you define a trap as being of one of the four types: DATA, IO, GT, and TTABLE. For the format and additional information on the ID operand, refer to the *VM/SP CP System Command Reference.*

Directing Your CPTRAP Output

CPTRAP output is collected in the CPTRAP spool file (CPTRAP FILE) spooled to a user on the system. If the CPTRAP TO operand is not issued, the file is spooled to the invoker of the CPTRAP command. Use the TO operand if you want the file to be sent to a virtual machine other than the invoker of the CPTRAP command.

DASD Space Considerations for CPTRAP

The DASD space consumed by CPTRAP will continuously grow unless you specify WRAP with the TO operand. You should decide how many pages of DASD should be dedicated to tracing and supply that number with WRAP. At least 16 pages must be specified.

A *wrap* spool file reuses spool space. After the number of 4K records that you indicated have been collected, new CPTRAP records overlay the older records already in the file. This allows you to limit the total amount of spool space used by CPTRAP.

A *non-wrap* spool file does not reuse spool space. The spool space used by the file is limited to the spool space available on the system. When the CPTRAP file has filled 15Mb of spool space, the file is closed automatically and a new file is opened. Since CPTRAP records are not overlaid in a non-wrap file, the available spool space can be used very quickly if you do not choose the entries selectively.

The data collection rates for CPTRAP could be over 1Mb per second on a heavily loaded system. At this rate, an entire 3380 DASD can be filled in between eight and nine minutes. Failure to use the WRAP operand under these conditions will result in system spool space being filled rather quickly. You should decide how much DASD space should be devoted to the CPTRAP information before the CPTRAP command is invoked and specify WRAP to limit spool space consumption.

You can also limit the amount of CPTRAP data recorded if you are recording trace table information. You do this by filtering the trace table entries recorded using the

INTABLE and INFILE operands of the TTABLE type traps. With these operands, you can limit the amount of CP trace table data being recorded.

Getting Tracing Started

Use the ENABLE operand of CPTRAP to start tracing after you have fully defined the traps. You can enable traps individually or by sets if you group related traps into sets. You can also enable all defined traps using the ENABLE ALL operands. After you issue ENABLE, CPTRAP collects data associated with a trap in the CPTRAP file.

Turning Off Tracing

To turn tracing off while preserving all trap ID definitions, use the DISABLE operand of CPTRAP. After you disable a trap, no more records that correspond to that trap are collected. Disabling the last enabled trap results in an automatic close of the CPTRAP file.

Altering Tracing

To alter tracing:

- Enable existing traps by trap ID or trap set
- Disable existing traps by trap ID or trap set
- Define new traps
- Redefine existing traps.

If you want to alter the parameters associated with a particular trap ID, the trap ID must be disabled before you can alter it unless it is a type TTABLE trap. You do not have to disable type TTABLE traps to alter them.

Analyzing Data

CPTRAP places the data collected in the CPTRAP spool file CPTRAP FILE which is spooled to the user specified as the receiver. You can access this file through an IPCS session. For more details, see the *VM/SP Interactive Problem Control System Guide and Reference*.

Ending Tracing

If trap IDs are no longer needed, you can issue CPTRAP with the DROP operand to remove them. Note that issuing DROP ALL erases all defined trap IDs. To end a CPTRAP data collection session, issue the CPTRAP STOP command. This command disables and drops all existing trap IDs and closes the CPTRAP spool file if it exists.

CPTRAP Command

You can use the privilege class C CPTRAP command to define a trap or set of traps, to start and stop tracing, and to provide other services associated with the CPTRAP facility. For details on the format and operands of this command, refer to the *VM/SP CP System Command Reference*.

CPTRAP operands can be viewed as belonging to one of the following groups:

- Definition
- Destination
- Activation
- Display.

Use the CPTRAP definition operands (ID operand) to identify a tracing condition with a name (trap ID) and optionally with a set of other traces (trap set). You also use definition operands to assign one of the four type traps (IO, TTABLE, DATA, and GT) to a trap ID.

Use the destination operands to direct CPTRAP output to the CPTRAP spool file (CPTRAP FILE). With these operands, you can direct the spool file to another user ID or to yourself. You can also specify WRAP to conserve spool space. Use CLOSE to end recording to a CPTRAP spool file and start another file.

Use the activation operands to start or end the tracing of individual trap IDs and of trap IDs in trap sets. These operands also allow you to erase all CP knowledge of individual trap IDs and trap IDs in trap sets.

Use the DISPLAY operand to look at the CPTRAP status of each trap ID or trap set.

Recording I/O Activity in the CPTRAP File

To trace all I/O activity on a specified real device or range of devices, use the TYPE=IO operand. The times, channel programs, CSWs, and data transferred for each I/O operation to the specified device or devices is made available for data analysis. You can enable up to 255 IO type trap IDs at a time.

Collecting I/O Activity Entries in the CPTRAP File

To collect I/O activity entries, do the following (some steps are optional):

1. Define one or more trap IDs as type IO.
2. Optionally, direct the CPTRAP output to a user ID on the system. If you do not specify a user ID, the CPTRAP output is spooled to the user ID that invokes the CPTRAP commands. To conserve DASD space, use the WRAP operand.
3. Start tracing by using the CPTRAP ENABLE operand.
4. Optionally, alter tracing by either defining and enabling new trap IDs or disabling and enabling existing trap IDs.
5. Turn off tracing (after a period of time) by using the CPTRAP command with the DISABLE operand.

Example: The following sequence of commands illustrates how to define, enable, and disable two trap IDs of type IO. These trap IDs, IO1 and IO2, are put into trap set IOSET. The destination is specified as a spool file for user ID "USER2" with a wrap size of 100.

```
cptrap id io1 set ioset type io dev c41-c47 iodata 200

cptrap id io2 set ioset type io dev a81-a87 iodata 400 user user1

cptrap to user2 wrap 100

cptrap enable set ioset

cptrap disable id io1
   (*This shows that IO1 can be disabled while IO2 is running)

cptrap disable id io2
```

In the next example, assume that users on your system complain of CMS abends that occur for no apparent reason. You discover that the users complaining have the same copy of CMS in common. This copy of CMS is on a 3380 at address 280. You also discover that CMS is abending because of program check code 1 errors. You suspect that some of the text decks comprising CMS are corrupted, so you regenerate that copy of CMS and the problem persists. The only explanations you have now are that the page causing the abends is corrupted during the transfer into main storage or that the users' segment or page tables are being corrupted. To define traps that would give more accurate data to determine which of these theories is correct, issue the following CPTRAP commands:

```
cptrap id i1 type io dev 280

cptrap id d1 type data loc (hexloc) xxxx dl g11+10%.40

cptrap id d1 dl g11+10%%.20 g11+10%+4.40 (... for x page tables)
```

The first CPTRAP command defines an IO trap, I1, for device 280.

The second CPTRAP command defines a data trap, D1, at the entry to the dispatcher. The data to be traced is the VMSEG in the segment table. (For additional information on DATA type traps, see "Recording CP Data in the CPTRAP File" on page 110.)

The third CPTRAP command specifies the real page address (g11 + 10%%.20) and the next segment table entry (g11 + 10% + 4.40).

Type IO Entries in the CPTRAP File

CPTRAP uses a X'3C' trace entry to construct the type IO entry that it places in the CPTRAP file. This entry contains the following information:

- Trap ID of the trap that created this entry.
- Trap set that this trap ID is a member of.
- Trap type of the trap ID that created this entry. (This is an 8-byte character field.)
- Name of routine to format this entry (used by IPCS).

The X'3C' IO trace entry contains one or more subsections, each composed of a CCW, an IDAW (Indirect Data Addressing Word) list (if there are IDAWs for this CCW), and CCW data.

The fields in this section are:

- User ID that I/O is being traced for
- Device being traced
- Number of bytes being traced for each CCW
- Flag byte. The following bits are defined:

Bit	Meaning
X'80'	Data truncated in this entry.
X'40'	Unsolicited interrupt

- Address portion of I/O old PSW.

The following fields are repeated for each CCW in the program. For reads and writes, the data is also traced.

- Channel status word at interrupt time
- First word of CCW being traced
- Second word of CCW being traced
- Real address of this CCW.

The following fields are used if the IDA bit is off in the CCW:

- Length of data for the CCW
- Variable length field that contains the data that was traced for the CCW (always ends on a word boundary).

The following fields are used if the IDA bit is on in the CCW:

- Count of IDAWs for this CCW
- List of IDAWs
- Data fields (up to 2K), one for each IDAW in the list, that contains the data traced for the IDAW.

For the format of the X'3C' trace entry, refer to *VM/SP CP Data Areas and Control Blocks*.

Recording CP Trace Table Entries in the CPTRAP File

To control which CP internal trace entries are collected in the CPTRAP file and the internal trace table itself, use the TTABLE operand. The trace codes that you specify for the INFILE and INTABLE operands correspond to the typenums specified for VM/SP releases prior to Release 6. Refer to Figure 5 on page 75 for the format of the CP trace table entries. Because of the nature of trace table recording, you can enable only one trap ID of type TTABLE at any one time, even though more than one may be defined. However, you can dynamically alter the definition of the enabled trap ID.

Note: The MONITOR command can override the INTABLE operands. Also, OPTIONS COPY in your local MACLIB can override INTABLE settings.

CP maintains the internal CP trace table in real storage. It is a "wrap" table that continuously overlays previously stored information with new trace table entries. As

a result, all the information needed to determine the cause of a problem may not be present in the trace table. CPTRAP allows you to record selected CP trace table entries in a spool file. Thus, you can save CP trace table entries that would be lost when the internal trace table wraps.

Collecting Trace Table Entries in the CPTRAP file

To collect CP trace table entries, do the following (some steps are optional):

1. Define one or more trap IDs as type TTABLE.
2. Select the appropriate selectivity (that is, select which trace entries are to be recorded in the internal trace table and the CPTRAP spool file).
3. Optionally, direct the CPTRAP output to a user ID on the system. If you do not specify a user ID, the CPTRAP output is spooled to the user ID that issues the CPTRAP command. To conserve DASD space, use the WRAP operand.
4. Start tracing by using the CPTRAP ENABLE command. Once you do this, the selected CP trace table entries are moved from the internal trace table to the CPTRAP spool file without changing their format or length.
5. Optionally, alter tracing by either defining new trap IDs, disabling and enabling existing trap IDs, or redefine selectivity on the enabled TTABLE type trap ID to change selectivity while CPTRAP is running.
6. Turn off tracing (after a period of time) by using the CPTRAP DISABLE command.

The following sections discuss steps 2 through 4 in more detail.

Specifying Selectivity: You can select the CP trace table entries collected in the spool file by trace type (typenum) using the INTABLE and INFILE operands. For a list of the defined trace types see Figure 5 on page 75. CPTRAP allows you to further select input trace table entries on the INFILE operand. The three allowed fields are VMBLOK address (VMBLOK), real or virtual device address (DEVADDR), and various code fields (CODE). Note that this selectivity is only available on INFILE typenums. See the *VM/SP CP System Command Reference* for more details on the CPTRAP command and its selectivity options.

INTABLE lets you define which CP trace entries are recorded in the CP internal trace table. INTABLE values that you define on CPTRAP commands correspond to bits defined by the field TRACEFLG at location X'400' in the PSA. Some of these bits control the trace table recording of more than one CP trace table entry. Thus, turning on and off some INTABLE typenums affects the recording status of other CP trace entries. The following table shows groups of typenums whose recording is controlled with the same bit:

Group	Typenums	Description
1	05 19	I/O interrupt Simulated I/O interrupt
2	06 07	Obtain free storage Return free storage
3	0B 0E 0F 14 18 1B 21	Start I/O Test I/O Halt Device Clear Channel Start I/O Fast Release Clear I/O Test Channel

The following examples show the use of these typenum groups using a previously defined and enabled TTABLE trap ID called t1:

Command	Result
`CPTRAP ID t1 INTABLE 07 OFF`	Both 06 and 07 are not recorded in the internal trace table or the CPTRAP file
`CPTRAP ID t1 INTABLE 06 ON 07 OFF`	Both 06 and 07 are not recorded in the internal trace table or the CPTRAP file
`CPTRAP ID t1 INTABLE 19 OFF 05 ON`	Both 05 and 19 are recorded in the internal trace table and the CPTRAP file
`CPTRAP ID t1 INTABLE 0B OFF 0E ON`	0B, 0E, 0F, 14, 18, 1B, and 21 are recorded in the internal trace table and the CPTRAP file
`CPTRAP ID t1 INTABLE 0B ON 0E OFF`	0B, 0E, 0F, 14, 18, 1B, and 21 are not recorded

No TTABLE operands are required before you enable a TTABLE trap ID. However, the default values of a TTABLE type trap ID are INTABLE ALL ON and INFILE ALL OFF. This means that nothing will be collected in the CPTRAP file.

You must enable a TTABLE trap ID (with at least some INFILE typenum selectivity turned on) before any CP trace table entries are recorded in the CPTRAP file.

Filtering: You should filter trace entries to reduce the probability of data loss. Trace types eliminated with the INTABLE operand during the enable function for a trap are not entered in the internal trace table and are not available in dumps of CP.

Use the INFILE operand to define the trace entry types to be recorded in the CPTRAP spool file. Only use INFILE if the trace entry types you want in the CPTRAP spool file are a subset of those being entered in the internal trace table.

Use the INTABLE operand to define which types of CP trace entries are to be recorded in the internal trace table. The default is that all defined trace entry types are included. Any trace entry type that you excluded with the INTABLE parameter is automatically excluded from the CPTRAP spool file.

For best performance, use INFILE ALL ON. Limit the use of specific INFILE operands to situations where a trace entry type is highly needed in the system dump but is not wanted in the CPTRAP spool files because this will increase the amount internal processing required and possibly result in lost data.

You cannot filter trace entries with the INTABLE operand according to device, code, or VMBLOK. However, you can do this with the INFILE operand.

For Release 6, filtering of trace codes X'3D', X'3E', and X'3F' has been changed from previous releases.

X'3D' and X'3E' entries apply only to GT type traps and cannot be controlled using TTABLE traps. No message is issued if X'3D' and X'3E' are specified for TTABLE traps.

X'3F' entries are supported under the type trap of TTABLE although they are never entered in the trace table. Using the INTABLE operand has no effect on the recording of these entries. These entries must be controlled by using the INFILE operand.

Examples: The example below shows how two trap IDs of type TTABLE could be defined. The tt1 trap ID allows only 05, 06, 07, and 19 entries to be written to the trace table and allows only 06 and 07 entries to be written to the CPTRAP file. The tt2 trap ID allows all entries to be written to the trace table and the CPTRAP file.

Note: Only one trap ID of type TTABLE can be enabled at one time, but this trap ID can be altered while it is enabled.

The destination is specified as a spool file for the issuer with a wrap size of 4000.

```
cptrap id tt1 intable all off

cptrap id tt1 type tt intable 05 06 07 infile 06 07

cptrap id tt2 type tt intable all on infile all on

cptrap to * wrap 4000
```

To alter the trap with trap ID tt1 after it has been enabled, you could issue the following sequence of commands. Note, however, that tt1 must be disabled before tt2 can be enabled. This is because two tt type traps cannot be enabled at the same time.

```
cptrap enable id tt1

cptrap id tt1 intable 0b infile 05

cptrap disable id tt1

cptrap enable id tt2
```


Recording Virtual Machine Data in the CPTRAP File

You can record guest virtual machine events in the CPTRAP spool file by using type trap GT. Only one virtual machine or group can be enabled for each trap, and each virtual machine can only be enabled for one trap at a time. The entries recorded using this interface result in X'3E' trap records and X'3D' records. X'3E' records can be created for virtual machines whether or not they are a member of a group. X'3D' records can only be created for virtual machines that are members of a group. These records include the trap ID and trap set of the TRPBLOK associated with the request and data supplied by the virtual machine.

The virtual machine interface lets a virtual machine send data to CPTRAP to be added to the CPTRAP spool file. Any program running in VM/SP (for example, an application program, CMS, GCS, TSAF, SFS, PVM, and CICS/VM) can use the interface to have data written to the CPTRAP spool file. The data gathered in the CPTRAP file by this interface could help determine the problem in your program.

Collecting Entries in the CPTRAP File for Type GT Traps

To collect virtual machine entries in the CPTRAP file, do the following (some steps are optional):

1. Define one or more trap IDs as type GT, and optionally specify selectivity for the kind of virtual machine entry to be collected (X'3D', X'3E', or ALL) Use either the ALLOWID or GROUPID operand to enable the virtual machine(s) for tracing.
2. Optionally, direct the CPTRAP output to a user ID on the system. If you do not specify a user ID, the CPTRAP output is spooled to the user ID that invokes the CPTRAP command. To conserve DASD space, use the WRAP operand.
3. Make sure the program running in the virtual machine contains the virtual machine interface to CPTRAP.
4. Start tracing by using the CPTRAP ENABLE command on the trap IDs set up for the enabled virtual machines. Once this is done, CPTRAP can construct the virtual machine entries and put them into the spool file.
5. Optionally, alter tracing by either defining and enabling new trap IDs or disabling and enabling existing trap IDs.
6. Turn off tracing (after a period of time) by using the CPTRAP DISABLE command.

The following sections describe steps 1-4 in more detail.

Defining Traps and Specifying Selectivity: Use the CPTRAP definition operands to define a GT type trap:

- Use *trapid* to give the trap a name.
- Optionally, use the SET operand to identify this trap as a member of a trap set. A trap set can be used to group traps into functionally or logically related sets.
- Use the TYPE operand to specify the type as GT.
- Use the ALLOWID or GROUPID operand to enable the virtual machine to send data to the CPTRAP file. ALLOWID enables an individual virtual machine, and GROUPID enables all the virtual machines in a group that you specify.

If the virtual machine is not logged on or disconnected, an ALLOWID trap ID for that virtual machine cannot be enabled. If no virtual machines are enabled for a specific group, a GROUPID trap ID for that group cannot be enabled.

Any enabled ALLOWID trap ID for a virtual machine is disabled if a GROUPID trap ID is enabled and the virtual machine is a member of that group. That is, GROUPID tracing overrides ALLOWID tracing for individual virtual machines. Later, any new virtual machines entering the group are automatically enabled for tracing. A virtual machine remains enabled for tracing until it severs the connection to the group, logs off, or the CPTRAP trap ID is disabled.

- Use the 3D, 3E, or ALL operands to specify selectivity. The selectivity options are:
  - X'3D' for group virtual machine data
  - X'3E' for individual virtual machine data
  - ALL means that both X'3D' and X'3E' records can be recorded by virtual machine(s) covered by this trap ID.

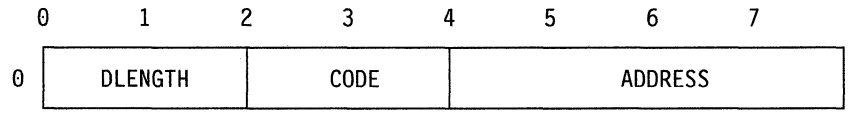
  If X'3D' entries are specified for an individual that is virtual machine not in a group, the trap can be enabled but no data is provided.

Setting Up the Virtual Machine Interface: The virtual machine interface to CPTRAP is a parameter list and a class 10 monitor call instruction. There is no restriction on the number of interfaces that may be active at one time, or on the number of virtual machines that can use them.

You can insert the interface into a program in two ways:

- Modify your program to include the interface, and then reassemble the program.
- Use the CP STORE command to store the interface into a program problem area.

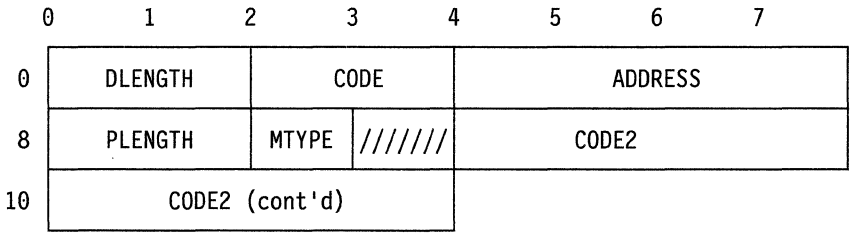
Set up register 1 with the address of an eight-byte parameter list that identifies the data to be included in the CPTRAP file. The format of the parameter list for monitor codes 0 and 1 is as follows:



Field	Contents
DLENGTH	Length of virtual machine data
CODE	Individualizing code that you assign
ADDRESS	Virtual address of data in virtual machine storage

The data cannot be longer than 2048 bytes, and must be in the virtual machine. If the length is greater than 2048 bytes, only the first 2048 are taken with no indication that the data has been truncated.

The format of the parameter list for monitor code 2 is as follows:



Field	Contents
DLENGTH	Length of virtual machine data. The data length is variable up to 2048 bytes.
CODE	Individualizing code that you assign
ADDRESS	Virtual address of the data to be included in the CPTRAP file. The data must be in the virtual machine that issues the monitor call instruction.
PLENGTH	Length of parameter list (in bytes). This must be at least X'C'. If the CODE2 field is specified, the field must be at least X'14'.
MTYPE	Code that identifies the type of virtual machine making the entry. The values are: • X'01' - TSAF virtual machine entries • X'02' - SFS server machine • X'03' - PVM • X'04' - CICSVM • X'05' through X'FD' - reserved for IBM use • X'FE' - Field engineering entries • X'FF' - User installation entries (USER1)
CODE2	Additional individualizing code information assigned by the user. The information is in EBCDIC format. If the parameter list is not long enough for this field, the corresponding field in the CPTRAP X'3E' record contains blanks. For X'3E' records not created using the MC 10,2 interface, this field contains blanks.

The individualizing code is used to look selectively at the virtual machine entry in the CPTRAP file when you use IPCS. This individualizing code will be present in each entry. If the individualizing code is unique for each interface that you set up, it will be easy to review data selectively in the CPTRAP file that came from a particular virtual machine interface.

A monitor call instruction can be executed in virtual supervisor or virtual problem state, BC mode or EC mode, and in multilevel environments. Multilevel is defined as VM/SP running a guest virtual machine of a VM/SP system.

The supported monitor codes are 0, 1 and 2. All other monitor codes are ignored and control returns to the invoker with no indication that the virtual machine data was ignored.

The format of the monitor call instruction is as follows:

MC x,10

x = 0 indicates that the data to be added to the CPTRAP file is general virtual machine data. Any virtual machine can use a monitor code 0.

x = 1 indicates that the data to be added to the CPTRAP file is virtual machine group data. Only a virtual machine that belongs to a group can use a monitor code 1.

x = 2 identifies an extended format for the parameter list that the virtual machine passes (general virtual machine data). Any virtual machine can use a monitor code 2.

The following chart shows the type of entry (if any) made in the CPTRAP file for the six possible situations that can arise:

	Virtual Machine is in a Group	**Virtual Machine is not in a Group**
Monitor Code 0 is issued	3E	3E
Monitor Code 1 is issued	3D	no entry is made
Monitor Code 2 is issued	3E	3E
Any other monitor code is issued	no entry is made	no entry is made

Example of Virtual Machine Interface for Type GT Trap: Two types of virtual machine data can be recorded in the CPTRAP file.

- *General virtual machine data* is sent by any virtual machine that is enabled and uses a monitor code 0 or 2 when passing the data to CPTRAP.
- *Group virtual machine data* is sent by a virtual machine that is enabled, uses a monitor code 1 when passing the data to CPTRAP, and belongs to a group.

One way that you might use the virtual machine interface to CPTRAP is to capture some data being changed incorrectly by the program. For example, you could include the following code at the appropriate location in the program:

```
         .
         .
         .
DOTHIS   CNOP  0,4            HERE TO RECORD ENTRY IN CPTRAP FILE
         STH   R2,DATALEN     PUT LENGTH OF DATA YOU WANT IN PLIST
         ST    R6,DATADDR     PUT ADDRESS OF DATA YOU WANT IN PLIST
         BAL   R1,AROUND      SET UP POINTER TO PLIST
*                             PARAMETER LIST:
DATALEN  DS    AL2              2 BYTES FOR LENGTH
         DC    AL2(5)           2 BYTES FOR CODE .. (THIS IS 5)
DATADDR  DS    AL4              4 BYTES FOR ADDRESS
AROUND   DS    0H
         MC    0,10           SEND GENERAL VM (3E) DATA TO CPTRAP
         .
         ◄existing code in program to continue doing something►
         .
         .
```

To use this trap in the program, the program must be reassembled. You must do whatever is required to run this new version of the program. Then, each time something causes the code in the trap to execute, the parameter list would be set up and control would go to CPTRAP.

You can set up any number of these traps in the code at the same time. By making the individualizing code unique in each case, you can review the virtual machine entries selectively in the CPTRAP file. In the example here, the virtual machine entry created is for general virtual machines (type X'3E') and has an individualizing code of 5.

Starting Tracing: Use the CPTRAP ENABLE operand to start tracing. You can only start tracing on those virtual machines that have been enabled using either the ALLOWID or GROUPID operand.

Collecting the Virtual Machine Data in the CPTRAP File: When CPTRAP is activated, the monitor code interface gives control to CPTRAP. If X'3E' entries are being collected, the data identified by the parameter list is recorded in the CPTRAP file.

To activate CPTRAP and collect only general virtual machine data, issue the following set of commands:

```
cptrap id trap1 type gt allowid user1 3e

cptrap enable id trap1
```

Now, whenever the code in your trap executes, an entry should be made in the CPTRAP file. You can cause the reader file to be created by issuing:

```
cptrap close
```

which will cause a reader file to be created and the following response:

```
RDR FILE 0003  TO  USER1   COPY 001   NOHOLD
Ready;
```

Virtual Machine Entries in the CPTRAP File: CPTRAP constructs the virtual machine entry that is placed in the CPTRAP spool file. A X'3D' trace entry is used to collect CPTRAP data for guest traces that issue Monitor class 10 code 1 instructions to collect CPTRAP data. The X'3D' entry, which identifies the group virtual machine entry in the file, contains the following information:

- Individualizing code
- Trap ID information
- User data

The individualizing code is the same code that you specified in the parameter list. The individualizing code is necessary to look selectively at the virtual machine entry in the CPTRAP file when you use IPCS. The length of the virtual machine entry in the CPTRAP file is variable. It includes the length of the virtual machine data plus the length of the header.

The X'3E' trace entry is used to collect trace data for guest traces that issue Monitor class 10 code 0 and code 2 instructions to collect CPTRAP data. This trace entry contains information from the virtual machine parameter list and trap ID information.

For the format of this trace entry, refer to *VM/SP CP Data Areas and Control Blocks*.

Examples: The example shown below depicts defining, enabling, and dropping two trap IDs of type GT. These trap IDs, GT1 and GT2, are put into trap set GTSET. The destination is specified as a spool file for user ID "USER3" with no wrap size.

```
cptrap id gt1 set gtset type gt allowid user2

cptrap id gt2 set gtset type gt groupid gcsgroup

cptrap to user3

cptrap enable id gt2
   (*gcsgroup must be defined to start this trap*)

cptrap enable id gt1

cptrap drop all gtset
```

For the next example, assume the GCS virtual machine is experiencing intermittent abends concerning free storage. To determine what CP events are occurring during these abends, issue the following commands:

`ETRACE FRE GET`	This command is required to trace GCS events; FRE and GET trace FREEMAIN and GETMAIN requests, respectively
`CPTRAP ID g1 TYPE GT GROUP gcs`	This defines a GT type trap
`CPTRAP ENABLE g1`	This starts tracing
`CPTRAP CLOSE`	

After tracing is complete, you can check the CPTRAP spool file to determine what storage is being acquired or returned just before the abend and determine which module(s) issued the requests.

Recording CP Data in the CPTRAP File

There are two methods in which you can record CP data in the CPTRAP spool file. The first, and preferred method, is to define a trap ID of type DATA. A second method, which was also available prior to Release 6, is to place CPTRAP interface code in the execution path of the CP code where the data is to be recorded. The two methods are discussed individually in the following sections. First defining a DATA type trap is discussed, and then the CP interface to CPTRAP is described.

Using a DATA Type Trap to Record CP Data

You can use the DATA type trap to dynamically define trace entries to record execution of most code paths in CP. The LOC operand specifies which instruction, when executed, will trigger recording of status information. The DL (datalink) operand defines the information to be collected for execution of that instruction.

CP trap IDs of type DATA allow you send information to be recorded in the CPTRAP spool file with a X'3C' trace entry. You can use the data collected to solve a problem in CP code.

Collecting CP DATA Entries in the CPTRAP file: To collect CP DATA entries in the CPTRAP file, do the following (some steps are optional):

1. Define one or more trap IDs as type DATA.
2. Using a CP LOAD MAP, find the virtual address of the instruction to be trapped.
3. Use the LOC operand for DATA to define the location and contents for the instruction to be traced.
4. Use the DL operand for DATA to define what debugging information should be collected when the instruction is executed.
5. Optionally, direct the CPTRAP output to a user ID on the system. If you do not specify a user ID, the CPTRAP output is spooled to the user ID that issues the CPTRAP command. To conserve DASD space, use the WRAP operand.
6. Start tracing by using the CPTRAP ENABLE command. Once this is done, CPTRAP can construct the CP entries and put them into the spool file.
7. Optionally, alter tracing by either defining and enabling new trap IDs or disabling and enabling existing trap IDs.
8. Turn off tracing when you think enough information is collected by using the CPTRAP DISABLE command.

Using the LOC Operand: Use the LOC operand to specify the hexadecimal virtual address of the instruction in the CP nucleus that defines the trap point at which the specified data is to be collected. If this address is in a pageable module, the module is automatically locked into storage.

There are certain modules in which you cannot define DATA type traps. These modules are restricted because enabling DATA type traps in them would cause recursive tracing or locking problems in the processing of CPTRAP data. If you try

to define a trap that violates one of these restrictions, the command is rejected and the message

```
DMKDTR369E Invalid location for trapping - command rejected
```

is issued. The restricted modules are:

DMKDTR	DMKTRO
DMKDTS	DMKTRP
DMKGTR	DMKTRT
DMKSVC	DMKTRV
DMKTCW	DMKTRX
DMKTCX	DMKTTR

Note: These modules are only restricted before entry point DMKxxxET.

Also, you cannot define locations that are not on halfword boundaries.

The *instruction* operand must be specified to allow CPTRAP to verify that the address you specified with *hexloc* contains the instruction you are defining the trap for. It is required for DATA traps before they can be enabled. It must be the hexadecimal representation of the contents of storage at the specified address. *Instruction* must be at least as long as the instruction being overlaid. If you specify *instruction* as hexadecimal data that is longer than the actual instruction, any extra data is ignored. If the *instruction* does not coincide with the storage contents, you will receive the following message:

```
DMKDTR349E String supplied does not match storage contents
```

and the command is ignored. The existing trap ID definition is not modified.

Using the DL Operand: You can use a datalink string, specified with the DL operand, to define the data to be traced. For the format, syntax, and examples of datalink strings refer to *VM/SP CP System Command Reference*.

You use these datalink strings to define how to combine constants and indirect specifications to reach the data to be traced. You can define data either by using the current register contents as pointers or by knowing the real storage address of the data to be traced.

CP DATA Entries in the CPTRAP File: CPTRAP uses the X'3C' trace entry for trap IDs of type DATA to construct the CP entry that it places in the CPTRAP spool file. The X'3C' entry for DATA type traps contains the following information:

- Individualizing code
- Trap ID that created this entry
- Trap set that this trap ID is a member of
- Trap type of the trap ID that created this entry
- Name of routine to format this entry
- Number of datalink strings in this entry
- Virtual address of trap point for this trap ID
- Bytes of trap generated data for this datalink string
- Datalink strings.

For the format of the X'3C' trace entry, refer to *VM/SP CP Data Areas and Control Blocks.*

Example: The example shown below illustrates how a trap ID of type DATA can be defined and enabled. The trap ID is called DSPLOOP, and it is defined at address 4B248. This address contains the instruction L R2,846(,R12) which assembles to object code 5820C846. The data link string specified is G4+8%.2. (G4+8%.2 means that the contents of GPR4 plus 8 point to a location that contains an address. That address contains the two bytes to be used.) The destination is specified as a spool file for the issuer with a wrap size of 2000.

```
cptrap id dsploop type data loc 4b248 5820c846

cptrap id dsploop d1 g4+8%.2

cptrap to * wrap 2000

cptrap enable id dsploop
```

Determining the Size of a Trace Record: A trace record collected for any DATA trap cannot exceed 4064 bytes including the header. Therefore, the amount of data collected cannot exceed 4000 decimal bytes. Also, the maximum number of datalinks allowed for any one trap ID is 255.

Use the following formula to determine the number of bytes collected:

38 + (3 * number of datalinks)
  + (sum of the length of the datalinks strings)
  + (sum of the requested data)

The value obtained by this formula must be less than or equal to 4064.

Here are some examples. Suppose you issue the following:

```
cptrap id t1 type data loc xxxx iiii
```

Since no data links have been defined, the number of datalinks = 0. Therefore, the formula gives the default, which is 38.

length = 38 number of datalinks = 0

Now if the following command is issued:

```
cptrap id t1 d1 g12.20
```

The number of datalinks becomes one, the length of the string is six, and the length of the data collected is 20. Substituting in the formula gives:

length = 38 + (3 * 1) + (6) + (20) = X'61'

Now if the following command is issued:

```
cptrap id t1 d1 440
```

The number of data links becomes two, the sum of the previous command's data link string plus the length of the data is 26, the length of the string for this command

is three, and the length of the data is four (four because no length was specified, four is the default). This gives the following result:

length = 38 + (3 * 2) + 26 + 3 + 4 = X'6B'

Thus, the total record size for the DATA trap produced by the previous set of commands is X'6B'.

Other Considerations: You should also consider the following when setting DATA type traps:

- Data is collected for trap IDs of type DATA before the overlaid instruction executes. If no datalinks are specified for the trap ID, a X'3C' entry is created in the CPTRAP file, but no data is collected.
- Since there is no way to guard against an overlay of a valid operation code somewhere other than at an instruction boundary, take care to match the instruction with the boundary. DATA traps should *always* be placed at instruction boundaries. Placing these traps at other locations causes unpredictable results and probable system abends.
- Datalink strings are evaluated in a left to right scan.
- Do not try to trap an instruction that is the target of an execute statement. This will cause unpredictable results. No error message is issued in this case. However, you can avoid this condition by tracing the execute instruction itself.
- If you make a typing error while entering the datalink string, use the DROP operand before redefining the trap ID. We recommend that you save the definitions of traps that contain long datalink strings in EXECs.
- In addition to the modules that would definitely cause recursive situations, some modules are invoked by CPTRAP after collecting a page of data. No restriction prevent you from placing traps in these modules, but if the records produced by these traps are one page long, recursion occurs and CPTRAP processing will loop. Therefore, we recommend that trap ID records for the following modules be well under a page long:
  - DMKIOS
  - DMKIOT
  - DMKPAG
  - DMKPAH
  - DMKRPA
  - DMKSTK
- Avoid placing traps in code paths that hold spin locks. Placing traps in these code paths will increase the path length under the spin lock and seriously degrade system performance.
- If interpretation of a datalink string during execution results in a negative or invalid address, a X'3C' error record is added to the CPTRAP file to indicate that the error occurred. However, you will receive no external indication that this error occurred.
- Specifying extra datalink strings for a defined but disabled trap results in addition to the definition.

- To trace data in the absolute PSA, reverse prefixing must be used. If interpretation of a datalink string results in an address in the prefix page of the executing processor, the data is obtained from absolute page 0.
- You can trace instructions in the PSA. If you enable a DATA trap in the PSA of an MP system, the instruction tracing occurs in both prefixed pages. Thus, the instruction is traced regardless of which processor the instruction is executed on.
- To set DATA traps in ECPS code paths, first issue SET CPASSIST OFF. If you do not issue SET CPASSIST OFF, any DATA traps in ECPS code paths are ignored, and no X'3C' entries are created for these trap IDs. For extended virtual machine assist code paths, use SET SASSIST OFF.
- Placing traps in frequently executed paths will have two detrimental effects on the system:
  - It will make a frequently executed path longer causing a system performance degradation.
  - Because the path is executed frequently, the data collection rate will be great, and the probability of data loss is increased.

Collecting CP Data By Including a CP Interface to CPTRAP: You can use a CP interface to CPTRAP to record CP information in the CPTRAP spool file. If you use this method, you will have to include the interface in the CP module you are tracing, either by using the CP STORE command or by reassembling that module and regenerating your system.

To collect CP entries in the CPTRAP spool file using this method, do the following (some steps are optional):

1. Place the CPTRAP interface in the CP code to be traced.
2. Define one or more trap IDs as type TTABLE.
3. Optionally direct the CPTRAP output to a user ID on the system. If you do not specify a user ID, the CPTRAP output is spooled to the user ID that invoked the CPTRAP command. To conserve DASD space, use the WRAP operand.
4. Specify selectivity for CP interface entries (X'3F' entries). (CPTRAP IO *trapid* TYPE TT INFILE 3F)
5. Start tracing by using the CPTRAP ENABLE operand. Once this is done, CPTRAP constructs the CP entries and puts them in the spool file.
6. Optionally, alter tracing by either defining and enabling new trap IDs or disabling and enabling existing trap IDs.
7. Turn off tracing (after a period of time) by using the CPTRAP command with the DISABLE operand.

Steps 1-5 are discussed in more detail in the following sections.

Setting Up the CP Interface: The CP interface to CPTRAP is a parameter list and a BALR 14,15 instruction. You can insert the CP interface into the CP code in two ways:

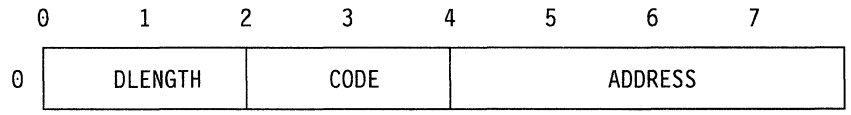
- Use the CP STCP command to store the interface into the problem area in the CP code.

- Modify the CP module to include the interface, reassemble the particular source module, and regenerate the system.

Be careful where you insert the interface in the CP code. There may be a condition code setting that has not yet been interrogated by the CP code. Any code inserted as part of the interface must not change that condition code setting. If the inserted code changes the condition code, it also must save and restore this setting. CPTRAP preserves the condition code setting in effect at the time the BALR 14,15 instruction executes.

You must set up register 15 with the address of TRAPOK in the PSA before the BALR instruction is issued. This is the address of the logic in module DMKPSA that determines whether CPTRAP is active. When the BALR 14,15 instruction is issued, register 14 gets the return address to the caller. The status of the CPTRAP facility determines what happens next. If CPTRAP is active, control goes to CPTRAP. If CPTRAP is not active, control returns to the caller immediately.

Set up register 1 with the address of an eight-byte parameter list that identifies the data to be included in the CPTRAP file. The format of the parameter list is as follows:



Field	Contents
DLENGTH	Length of CP data
CODE	Individualizing code
ADDRESS	Address of CP data to be added to the CPTRAP spool file

The data must be 2048 bytes or less, and must be in real storage. If it is larger than 2048 bytes, only the first 2048 are taken with no indication that the data has been truncated.

The individualizing code is used to look selectively at the CP entry in the CPTRAP file when you use IPCS. This individualizing code will be present in each entry. If the individualizing code is unique for each interface that you set up, it will be easy to review data selectively in the CPTRAP file that came from a particular CP interface.

You could use the CP interface to CPTRAP trace CP control blocks at various points in the CP code. For example, code in the module DMKQCN releases the storage used for CONTASKs. If you needed to record the information in a CONTASK for problem determination before it is released, you could include the following code at the appropriate location in DMKQCN:

```
         .
         .
         .
ALLDONE  DS    0H              HERE TO RELEASE THE CONTASK
         USING CONTASK,R6      GET ADDRESSABILITY TO CONTASK
         LH    R2,CONTSKSZ     GET CONTASK LENGTH IN DWORDS
         SLL   R2,3            CONVERT THE LENGTH INTO BYTES
         STH   R2,DATALEN      PUT LENGTH OF CONTASK IN PLIST
         ST    R6,DATADDR      PUT ADDRESS OF CONTASK IN PLIST
         USING PSA,R0          NEED ADDRESSABILITY TO PSA
         BAL   R1,AROUND       SET UP POINTER TO PLIST
*                               PARAMETER LIST:
DATALEN  DS    AL2               2 BYTES FOR LENGTH
         DC    AL2(3)            2 BYTES FOR CODE .. (THIS IS 3)
DATADDR  DS    AL4               4 BYTES FOR ADDRESS
AROUND   DS    0H
         LA    R15,TRAPOK      GET ADDRESS OF TRAPOK
         BALR  R14,R15         SEND DATA TO CPTRAP FILE
         .
         ◄existing code in DMKQCN to release the CONTASK►
         .
         .
```

Now, to use this trap in DMKQCN, you must reassemble the module and rebuild the system using the modified version of DMKQCN. Then, each time a CONTASK is released, a parameter list is set up and control goes to CPTRAP.

You can set-up any number of traps in CP code at the same time. By making the individualizing code unique in each case, you can review the CP entries selectively in the CPTRAP file. In the example, the CP entries created have an individualizing code of 3.

Specifying Selectivity: Use the CPTRAP INFILE operand to specify that CP entries are to be collected in the CPTRAP spool file. CP entries have a typenum of X'3F'.

Collecting the CP Data in the CPTRAP File: When CPTRAP is not active, control immediately returns to the caller, in the example we used earlier, DMKQCN. When CPTRAP is active, control is given to CPTRAP. If X'3F' entries are being collected, the data identified by the parameter list is recorded in the CPTRAP spool file.

To activate CPTRAP and collect only records that CP sends, issue the following set of commands:

```
cptrap id trap1 type tt infile 3f

cptrap enable id trap1
```

Now, whenever any CONTASK in the system is released, this is recorded in the CPTRAP file.

Suppose a user issued the following messages:

```
m op are you logged on today

m op did i catch this contask in the cptrap file?
```

This would have created two CONTASKS. When these are released, you would expect to find both of them in the CPTRAP file.

Stop the CPTRAP facility and create the reader file by issuing:

```
cptrap stop
```

When the CPTRAP file has been sent to your reader, you will receive the following response:

```
RDR FILE 0002  TO  USER1    COPY 001   NOHOLD
CPTRAP COMMAND COMPLETE
Ready;
```

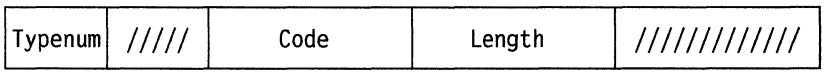
CP Entries in the CPTRAP Spool File: CPTRAP uses the X'3F' trace entry to collect CPTRAP CP interface data. This trace entry contains the following information:

- Individualizing code
- Trap ID that created this entry
- Trap set that this trap ID is a member of
- User data

The individualizing code that CP puts in bytes 2 and 3 of the trace entry is the same code that you specified in the parameter list. The individualizing code is necessary to look selectively at the CP entry in the CPTRAP file when you use IPCS.

For the format of the X'3F' trace entry, refer to *VM/SP CP Data Areas and Control Blocks*.

CP Entries in the CPTRAP File: CPTRAP constructs the CP entry that is placed in the CPTRAP file. An 8-byte header is appended to the front of the data that is passed by CP. The header identifies the CP entry in the file with the following format:



Disp	Field	Length	Description
0	Typenum	1	3F for CP data
1		1	Reserved byte
2	Code	2	Individualizing code
4	Length	2	Length of CP entry
6		2	Reserved

The individualizing code that CP puts in the third and fourth bytes of the header is the same code the user specified in the parameter list. The individualizing code is necessary to look selectively at the CP entry in the CPTRAP file when you use

IPCS. The length of the CP entry in the CPTRAP file is variable. It includes the length of the CP data plus eight for the length of the header.

Obtaining CPTRAP Status

To display status and selectivity information about the various traps that have been defined, use the class C QUERY CPTRAP command. For details on the syntax and operands of this command and the responses received, refer to the *VM/SP CP System Command Reference*.

To see the effects of issuing a specific CPTRAP command, use the DISPLAY operand on the CPTRAP command when you issue it.

Additional CPTRAP Considerations

Data Lost Situations

When you use the CPTRAP facility, data can be lost under the following circumstances:

- The CP trace table can wrap before the entries that have been made since the last recording can be written to a destination. This only happens if you have enabled a TTABLE trap ID with at least some INFILE selectivity on. If this occurs, you will receive the message:

  ```
  DMKTRT308I CPTRAP data lost
  ```

  and a X'3C' DATA LOST trace entry is created and sent to the destination. This entry marks where in the CPTRAP file the data was lost.

- When CPTRAP data is being recorded, it is kept in buffers temporarily while the I/O operation to DASD is being scheduled. If all the buffers are full and more data from the trace table needs to be written to them, the following message is sent to the invoker:

  ```
  DMKTRT308I CPTRAP data lost
  ```

  In addition, a DATA LOST trace entry is created and sent to the DASD destination as soon as a buffer becomes available.

A data lost message is issued when the system creates output faster than it can be transferred to the spool file. When this happens, the output file also indicates that data has been lost. The amount of data lost in any data lost situation is unpredictable. The possibility of a data lost situation is:

- Directly proportional to the rate of transfer of trace table data to spool.
- Directly proportional to the frequency and size of all interface data.
- Inversely proportional to speed of the spool DASD. This is a potential problem with the faster processors and/or with heavy use of the CPTRAP facility.

A reduced selection of trace types and CP or virtual machine data helps reduce lost data problems.

Checkpointing

Closed CPTRAP reader files are checkpointed in the same manner as any other spool files. In addition, if the system abends and the CPTRAP file is still open, the file is closed and checkpointed.

Running with Microcode Assist Active

The CPTRAP facility deactivates the dispatcher assists in ECPS:VM/370 to support the monitor call interface for virtual machines. Deactivation occurs only when the virtual machine being dispatched is enabled for tracing by a GT trap ID or when a TTABLE trap ID is enabled.

LOGOFF Considerations

The CPTRAP facility is stopped if the user who invoked CPTRAP logs off.

Spool Space Considerations

The CPTRAP facility is stopped if no spool space is available on the system. When the system is using 90% of its spool space, and again when it is using 100%, CP sends a message to the user. When no space is available CPTRAP closes the file, creates a READER file, and stops processing. It then issues the following message if any other CPTRAP commands are issued:

```
System spool space full; command rejected
```

CP/Virtual Machine Interface Errors

Two specific problems can occur in the following situations:

- Any byte of the parameter list or any byte of the data field lies outside of the virtual machine or CP storage due to an invalid address.
- An I/O error occurs while trying to read a page.

When these errors occur:

1. The system sends an informational message to the virtual machine user who started CPTRAP.
2. CP puts a special indicator, ADDR BAD, into the file.
3. The system ignores the data sent by VM or CP.

Release Level Conflicts and Migration Considerations

For Release 6:

- The structure of the CPTRAP command has been completely changed, you will need to learn the new syntax for doing the same functions that were available in previous releases. All EXECs and user programs that relied on the previous CPTRAP syntax may need to be changed.
- A X'20' time stamp entry appears in both the CP internal trace table and the CPTRAP spool file. Any programs that previously depended on trace table entry formats must be updated to recognize these time stamp entries.
- The QUERY CPTRAP command responses are enhanced and expanded.
- You can use RACF to control who can use the CPTRAP command.
- Because of the addition of the INTABLE option, the syntax and the use of CPTRAP typenums is changed.

- The CPTRAP reduction routine, TRAPRED, has been replaced by IPCS commands. The name of the DUMPSCAN command has been changed to IPCSSCAN and the PRTDUMP command has been changed to IPCSPRT. Any program or EXEC that invokes the command names from previous releases must be updated to invoke the Release 6 command names.

  Note: For migration, you can make copies of the IPCSSCAN and IPCSPRT modules and name the copies DUMPSCAN and PRTDUMP.

  For additional migration consideration regarding the elimination of TRAPRED and changes to IPCS, refer to the *VM/SP Interactive Problem Control System Guide and Reference*.

- Trace entries of the type X'3C' were added. Any programs written before Release 6 that depend on trace entry formats at reduction time must be updated to handle these new trace entries. Also, any program that created X'3C' entries must be updated so that it doesn't conflict with CP's entries.

- Trace entries X'3D', X'3E', and X'3F' were changed. Any programs that format these trace entries must be updated to process the new fields. Refer to *VM/SP CP Data Areas and Control Blocks*.

- Some functional characteristics of CPTRAP typenums are changed in relation to the INTABLE and INFILE operands of the TTABLE type trap. Previously, specifications of typenum selectivity following an ALL ON command implicitly turned off all other typenums not specified. With Release 6, this does not occur.

Displaying the CPTRAP Output

Refer to the *VM/SP Interactive Problem Control System Guide and Reference* for information on displaying CPTRAP output.

370X Dump Processing

The following sections discuss the Network Dump and the NCPDUMP. Use the NETWORK DUMP command to dump the 370x communications controller's storage. Use the NCPDUMP command to process CP spool reader files created by NETWORK DUMP command.

Network Dump Operations

This section only applies to 3704 or 3705 communication controllers that have been loaded by VM/SP. If you want to dump the contents of a 3725 or a 3705 that has been loaded by ACF/SSP, refer to *ACF/NCP V4, ACF/SSP V3 Diagnosis Guide*.

If 3704/3705 operations are erratic, fatal hardware errors occur, or some other internal error appears, the Communications Controller's storage should be dumped. The NETWORK DUMP command dumps the contents of 3704/3705 storage for NCP, PEP, or EP 3704/3705 control programs, if unit check or IPL required conditions are detected.

The format of the NETWORK command with the DUMP operand is:

NETWORK	**DUMP** *raddr* [**IMMED** / **AUTO** / **OFF**]

raddr
is the real hexadecimal address of the 3704/3705.

IMMED
is the default operand; it forces an immediate dump. The IMMED operand, if specified, does not reload the control program. Before 3704/3705 resources can be used again, the control program must be reloaded. To reload the control program after the NETWORK DUMP raddr IMMED command has executed, use the NETWORK LOAD raddr ncpname command.

If the IMMED operand is specified, a check is made to determine whether the "IPL required" sense status is present. If it is not, the following message occurs:

```
CTLR raddr IPL NOT REQUIRED; ENTER 'YES' TO CONTINUE
```

This pause in operations allows the operator an opportunity to check the NETWORK DUMP command line before engaging or terminating the operation.

AUTO
causes a dump if VM/SP subsequently detects a unit check condition or "IPL required" condition. If AUTO is specified, each time a dump is taken, the Communications Controller is reloaded with the 3704/3705 control program that was previously active.

OFF
resets a previously set AUTO (automatic dump) status.

Note: The dumps produced by the NETWORK command cannot be processed by the IPCSDUMP service program. NETWORK-initiated dumps are processed by the NCPDUMP (Network Control Program DUMP) service program created for this task.

NCPDUMP Service Program and How To Use It

NCPDUMP applies only to dump files that were dumped with the NETWORK DUMP command after the 3704/3705 was loaded by VM/SP.

NCPDUMP is a CMS command. It processes CP spool reader files created by 3705 dumping operations, that is, dump files that are produced as a result of the CP NETWORK command specified with the DUMP operand and either automatic or immediate mode.

The NCPDUMP file processing operation can include:

- Erasing a specific CMS NCPDUMP file after printing it
- Formatting the dump
- Printing the dump
- Assigning an identifier to the CMS NCPDUMP file
- Creating the CMS NCPDUMP file from the spool file.

Although NCPDUMP is a CMS command, its use is restricted to the user identified by the SYSDUMP operand of the SYSOPER macro in DMKSYS during VM/SP system generation. The operation of NCPDUMP is similar to IPCSDUMP operations. A general description of the NCPDUMP operation follows the command description.

The NCPDUMP command has the following format:

NCPDUMP	**[DUMP** *xx***][([ERASE][NOFORM][NCPBUFF][)]]**

DUMP*xx*
is the file name of a CMS file containing a 3704/3705 Communications Controller program dump. This dump was created by a previously invoked NCPDUMP command with the ERASE operand not specified.

ERASE
erases the current CP DUMP file or a specified DUMPxx (file name), saved CMS file.

NOFORM
specifies that a formatted control block is not desired.

NCPBUFF
specifies that a formatted listing of the NCP buffer pool is desired.

The NETWORK command invoked with the DUMPxx operand, as stated previously, produces CP files that contain the contents of a designated 3704/3705 Communications Controller unit buffer. These CP files reside as a spooled reader input assigned to a system designated user. The CMS NCPDUMP command invoked by this user formats (if requested) and prints the contents of these files.

The NCPDUMP program creates a CMS file with a file name DUMPxx and a file type of NCPDUMP, and erases the original spooled NETWORK initiated dump reader file. The created CMS file is erased if you specify ERASE; otherwise it is kept.

A maximum of ten dumped spooled files can be processed and saved, and later recalled, if necessary, by the system assignment of an xx identifier suffix to the CMS DUMPxx file name. The "xx" is a decimal number from 00 to 09, depending on any existing files of a similar name. For example, if the files DUMP00 NCPDUMP and DUMP01 NCPDUMP already exist, the new file would be called DUMP02 NCPDUMP. The file thus created is retained for later use unless the ERASE operand is specified, in which case the file is erased immediately after the dump is printed.

Stand-Alone Dump Facility

Overview

With the stand-alone dump facility, you can dump up to 16 Mb of real storage when VM/SP cannot create a CP Abend dump. This facility dumps all resident pages, CP and non-CP. The stand-alone dump facility cannot dump virtual machine storage and non-resident pages from the paging device.

To use the stand-alone dump program to dump the real storage, you must have access to IPL the real machine. You can IPL the stand-alone dump program from tape or DASD and direct the output to tape or printer. When using tape as the output device, reserve the complete tape for the stand-alone dump facility. Basic error recovery is available for DASD, tape, and printer 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 facility:

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

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

The stand-alone dump program communicates with the user with PSW wait codes. Refer to *VM/SP System Messages and Codes* in "Stand-Alone Dump Facility Wait State Codes." Once the CPU has gone into a wait state, the user can display the PSW, using conventional means, to find if the dump was successful.

The starting and ending addresses of the CP internal trace table are stored in the PSA at locations X'7B0' and X'7B4', respectively, in addition to the PSA locations X'0C' and X'10'.

Refer to Appendix C, "Stand-Alone Dump Formats" on page 253 for information on dump formats.

Devices that You Can Use to IPL Stand-Alone Dump

The following are the devices you can use to IPL the stand-alone dump program:

DASD	Tape
3330	2401
3333	2415
3340	2420
3344	3410
3350	3420
3375	3422
3380	3430
	9347

Notes:

1. If a DASD is selected as the IPL device:
   a. It cannot be the resident system device
   b. It must be CP formatted
   c. Cylinder 0 must be allocated as permanent space
   d. Cylinder 0 will be used.
2. The stand-alone dump IPL tape can be the same as the tape you direct the dump output to.
3. Do not try to IPL from a device that is not in the above list.

Devices to which You Can Send Dump Output

The following are the devices to which the dump output can be directed:

Tape	Printer
2401	1403
2415	1443
2420	3211
3410	3203 (Models 4 & 5)
3420	3289E (Model 4)
3422	3262 (Models 1, 5, & 11)
3430	4245
9347	4248

Notes:

1. You can specify a maximum of eight real addresses for the dump output device.
2. Do not mix printers and tapes in the same list. If you use a printer as the output device, the FCB should match the forms loaded in the printer. If the FCB does not match the form, data may be lost when the printer runs out of paper.
3. When you configure the stand-alone dump facility, you can use any printer type or tape type from the list of supported devices.

   For example, the SADUMP exec prompts you for the output device type with the following:

   ```
   PLEASE ENTER ONE OF THE FOLLOWING OUTPUT DEVICE TYPES:
   PRINTER:  (1403, 1443, 3203, 3211, 3262, 3289, 4245, 4248)
   TAPE:  (2401, 2415, 2420, 9347, 3410, 3420, OR 3430)
   ```

   If you specify 3420, the system expects the output to be directed to a tape device.

   The system will then request the output device address with the following:

   ```
   PLEASE ENTER REAL OUTPUT DEVICE ADDRESS OR LIST ADDRESSES
   (MAXIMUM OF 8) FOR TAPE:
   ENTRIES IN A LIST MUST BE SEPARATED BY A MINIMUM OF ONE BLANK.
   ```

   You must then respond with the address or list of addresses of the tape device(s) which can receive the output. Be sure the output addresses match the device type (tape in this example); otherwise, results are unpredictable. Keep in mind

that the generated stand-alone dump program does not check the address of the device for validity.

For VM/SP, you can enter up to three digits for the real output device address.

4. The stand-alone dump must have channel 1 defined in the FCB or carriage control tape, if the output device is a printer.
5. Do not send the stand-alone dump output to a device that is not included in the above list.

When you specify tape as the output device, the stand-alone dump program selects, as the dump output device, the first available device in the list, excluding the IPL device (if it is in the list). If you want the stand-alone dump output to go to the IPL tape, make all other devices that are in the list not ready. If no other device within the output address list is available and the IPL tape address is in the list, the IPL device will receive the dump.

If you select a tape for the dump output device, other than the IPL tape, the stand-alone dump facility:

1. Rewinds the tape to ensure that the dump is at the beginning of the tape
2. Sets the density to the highest value for the tape device.

If the tape device selected is the one on which the stand-alone dump facility resides, the facility will write the dump at the same density as the stand-alone dump program was written.

When using tape, reserve the complete tape for the stand-alone dump program; do not put the stand-alone dump program on a tape with the other stand-alone utilities. If you do not want to use the IPL tape as the dump output tape, you may want to put the stand-alone dump program on a mini-reel to make better use of tape resources.

If you select tape to be the output device type, use a single-volume, non-labeled tape for the stand-alone dump program. Be sure that the tape is non-labeled, because the facility does not check to ensure that it is a non-labeled tape.

You can issue the SPTAPE command with the SADump option to move the data from the output tape to a class V reader spool file which is IPCS compatible. To use SPTAPE command when dumping to the IPL tape (that is, if the IPL device address is the same as the dump device address), remember that the tape must be moved to the first tape mark. This tape mark identifies the beginning of the dump. From that point on, you can invoke IPCS to handle the stand-alone dump.

Stand-Alone Dump Program Generation

Your installation can generate the stand-alone dump program to customize the facility to your system configuration. This gives you control over the device used to IPL the stand-alone dump program and the output device for the dump. Invoke the SADUMP EXEC in a CMS virtual machine to do the generation.

Do not call the SADUMP EXEC from within another EXEC. Also, do not queue up the answers ahead of time when running the SADUMP EXEC. To generate a stand-alone dump program, enter "sadump."

To use the SADUMP EXEC:

- You must have read/write access to file mode A.
- The following files must exist on an accessed file mode:
  - DMKSP CNTRL (this is the default if no control is entered)
  - DMKLD00E LOADER
  - LDT DMKSADWT
  - DMKSAD TEXT.

You are asked to answer a series of questions that describe the environment where the stand-alone dump program will run. The SADUMP EXEC checks all input for validity, and returns messages if you enter invalid data. An example of the prompts and replies that appear on the virtual machine console during SADUMP EXEC execution is shown in "Example for Configuring the Stand-Alone Dump" on page 128.

Following the data that you provide, the SADUMP EXEC does one or more of the following:

- Creates a file with a name of SADGEN[4] ASSEMBLE, and places the file on the user's file mode A. The file has the SAD MACRO with the selected parameters.
- Assembles the SADGEN file to create the SADGEN TEXT file.
- Places the stand-alone dump program in the user's virtual card reader to be IPLed as desired. When the virtual reader is IPLed, the stand-alone dump program will be written on the IPL device.

Notes:

1. The real device address from which the stand-alone dump program is IPLed is not necessarily the same device address used when it was created.
2. It is impossible to verify the dump output address(es) and type at stand-alone dump generation time.

Using the Stand-Alone Dump Facility

To use the stand-alone dump facility:

1. Configure the stand-alone dump program.
2. Take the stand-alone dump.
3. Process the stand-alone dump from tape to a spool file.

Configuring the Stand-Alone Dump

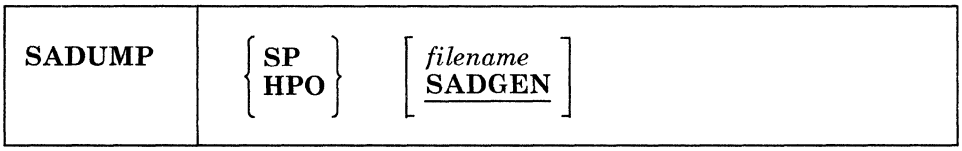
Before you can use the stand-alone dump program, you must configure the facility. This lets you configure the IPL device and the dump output device(s) for the stand-alone dump facility to match the real I/O.

Use the SADUMP EXEC during configuration to create and assemble the SADGEN ASSEMBLE file. The SADUMP EXEC places an IPLable deck in the virtual card reader. If the system detects an incorrect response, the exec gives an

[4] SADGEN is the default. The file name will be the same as specified in the SADUMP command if you do not use the default.

error message to you and requests a new response. For assembly errors, the exec will exit. If the stand-alone dump IPL device is a DASD, it must be CP formatted and the facility will use cylinder 0. Allocate cylinder 0 as permanent space.

The format of the SADUMP command line is:



SP
specifies that the stand-alone dump is going to be used on a VM/SP system, which will allow up to a 3-digit dump output real address.

filename
is the file name of the ASSEMBLE file that has the SAD MACRO. SADGEN ASSEMBLE is the default if you do not supply an operand. (Comply with the CMS guidelines for file names if a file name is specified).

The following items apply to configuring the stand-alone dump:

1. If the system generated more than one stand-alone configuration, use unique names for each configuration. The default is SADGEN. If you answer "Y" to replace one that already exists (refer to "Example for Configuring the Stand-Alone Dump" on page 128), the original is erased.
2. If you respond with "N" to any of the questions (refer to "Example for Configuring the Stand-Alone Dump" on page 128), the exec will go directly to the next question without doing the indicated work.
3. When the exec asks you to enter the real output device address, you are limited to three digits for VM/SP.
4. When the exec asks you to enter the control file, if you hit ENTER the exec uses the default of DMKSP.
5. The stand-alone dump configuration deck that the system puts in the virtual card reader is a class D file. You must place the deck in front of all the class D reader files before IPLing.
6. You must IPL the stand-alone dump reader file within a virtual machine. After the IPLable deck is in your reader, perform the following instructions:
   a. SET ECMODE ON
   b. SPOOL 00C CLASS D
   c. System CLEAR
   d. IPL 00C

   Note: Before you IPL the virtual reader, make sure that the IPL device is mounted and ready. If the IPL device is a tape, make sure the write ring is in.

Example for Configuring the Stand-Alone Dump

The following is an example of a stand-alone dump facility generation. In this example:

- You are placing the stand-alone dump program onto a 3330 device with an address of 150.
- The control file in this example is DMKSP. You may enter any control file you would like to use or take the default of DMKSP.
- The system will send the dump output to the first available 3420 tape drive whose address is 570, 571, 572, 573, 574, 575, 576, or 577.
- The file name of the file that has the SAD MACRO defaults to SADGEN.

Note: In the following example, "= = = = =>" indicates data that you enter.

```
    = = = = > SADUMP

The SADUMP EXEC:

   - OPTIONALLY CREATES A NEW SADGEN ASSEMBLE FILE CONTAINING
     A SAD MACRO WITH SELECTED PARAMETERS ON YOUR FILMODE A.

   - OPTIONALLY ASSEMBLES THE SADGEN ASSEMBLE FILE.

   - OPTIONALLY PLACES A SADUMP CONFIGURATOR DECK INTO THE
     VIRTUAL CARD READER.

NOTE: YOU MAY EXIT FROM THIS EXEC BY ENTERING 'QUIT' FOR ANY
RESPONSE.

DO YOU WANT TO CREATE A NEW SADGEN MODULE? (Y|N)
    = = = = > Y

PLEASE ENTER THE VIRTUAL DEVICE ADDRESS TO WHICH THE SAD
PROGRAM WILL BE WRITTEN (IPL DEVICE):
    = = = = > 150

PLEASE ENTER ONE OF THE FOLLOWING IPL DEVICE TYPES:
DASD:  (3330, 3333, 3340, 3344, 3350, 3375, 3380)
TAPE:  (2401, 2415, 2420, 9347, 3410, 3420, 3422, or 3430)
    = = = = > 3330

PLEASE ENTER ONE OF THE FOLLOWING OUTPUT DEVICE TYPES:
PRINTER:  (1403, 1443, 3203, 3211, 3262, 3289, 4245, 4248)
TAPE:  (2401, 2415, 2420, 9347, 3410, 3420, 3422, or 3430)
    = = = = > 3420

PLEASE ENTER REAL OUTPUT DEVICE ADDRESS OR LIST ADDRESSES
(MAXIMUM OF 8) FOR TAPE:
ENTRIES IN A LIST MUST BE SEPARATED BY A MINIMUM OF ONE BLANK.
    = = = = > 570 571 572 573 574 575 576 577
```

```
DO YOU WANT TO ASSEMBLE SADGEN NOW? (Y|N)
    = = = = > Y
ENTER THE CONTROL FILE YOU WANT TO USE.
THE DEFAULT IS DMKSP.
    = = = = > DMKSP

THE SADGEN MODULE IS NOW BEING ASSEMBLED.
DO YOU WANT TO PLACE AN IPL'ABLE DECK IN YOUR VIRTUAL
CARD READER? (Y|N)
    = = = = > Y

AN IPL'ABLE DECK EXISTS IN YOUR VIRTUAL CARD READER
IN CLASS D.  IPL THE READER TO PLACE THE STAND-ALONE
DUMP PROGRAM ON THE IPL DEVICE.
Ready;
```

Note: After you IPL your reader, if no errors occurred, you will receive a wait state code of 912.

Taking a Stand-Alone Dump

If you plan to dump 16 Mb of storage, use a tape density of 1600 or 6250 BPI. A 16 Mb dump may not fit on a tape at 800 BPI.

To invoke the stand-alone dump:

1. For multiple processor systems, stop all tightly-coupled processors. Do NOT clear storage.
2. For multiple processor systems, select the processor with the I/O configuration that has access to the resident volume address and the output device address(es).
3. Display locations X'0' to X'B' at the console. The stand-alone dump IPL sequence overlays these bytes, so they cannot be recovered.
4. Do a STORE STATUS operation on the CPU where you will IPL the stand-alone dump program. If you do not do the STORE STATUS, the following will not be saved in low storage:
   - CPU Timer
   - Clock comparator
   - Current PSW
   - Prefix
   - Model dependent features
   - Control registers
   - Floating point registers
   - General purpose registers.

   If the prefix value is not saved in low storage, the information from the prefix page is not available for the formatted section of the dump.
5. Mount and ready the volume that has the stand-alone dump program. If this is a tape, be sure to have the write ring in place.
6. Ready the output device, either tape or printer. If you want the system to place the stand-alone dump on the IPL tape, make all other tapes listed as output

devices (at generation time) NOT ready. If you do not want the stand-alone dump on a device that is listed as a possible output device, the device must not be ready at the time you IPL the stand-alone dump program.

7. IPL the stand-alone dump program. The stand-alone dump program will initially write the first nine pages of storage to the IPL device. This will provide an area to load the stand-alone dump program and work space. (See "Tape Format" on page 253 and "DASD Format" on page 255 for information about the DASD or tape format.) This step causes the system to place the dump on the output tape or the printer. (See "Tape Format" on page 253 and "Printer Format" on page 256 for information about the format of the output.)
8. When the system enters the wait state, display the PSW. A wait state of 912 indicates successful completion of the stand-alone dump. If the stand-alone dump program is unsuccessful because of some error that you can fix (for example, an unrecoverable I/O error on the output tape):
   a. Correct the error.
   b. Invoke a hardware RESTART to restart the stand-alone dump. (For example, type in RESTART on the appropriate panel on a 4300 processor.)

   If you re-IPL the stand-alone dump facility again, part or all the first nine pages of storage will be invalid. After the initial IPL, you cannot change the IPL address or IPL volume.

Processing the Stand-Alone Dump Data on Tape

If you directed the output to tape, re-IPL VM/SP. Then issue the SPTAPE command with the LOAD raddr and SADUMP operands to create an IPCS compatible spool file. This is the only way to transfer the data. After the system has created the spool file, enter the IPCS command, IPCSDUMP, to process the stand-alone dump. For more information on CP Abend Dumps see "Reading CP Abend Dumps" on page 78.

Chapter 4. Debugging CMS

This section describes the debug tools that Conversational Monitor System (CMS) provides. These tools can be used to help you debug CMS or a problem program. In addition, a CMS user can use the Control Program (CP) commands 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 CP and CMS commands described in previous chapters are:

- PER and ADSTOP, which set breakpoints (address stops) that stop program execution at specific locations.
- TRACE, which traces specific virtual machine activity and records the results on the terminal or printer.
- DISPLAY, which displays the contents of:
  - Channel Address Word (CAW)
  - Channel Status Word (CSW)
  - Old Program Status Word (PSW)
  - General purpose registers (GPR)
  - Virtual storage

  at the terminal.
- STORE, which:
  - Changes the contents of the control words (CAW, CSW, and PSW) and general purpose registers
  - Stores data in virtual storage locations.
- VMDUMP, which dumps virtual storage in a different format than the DUMP command; the output produced by VMDUMP can be processed by IPCS.
- DUMP, which dumps all or part of virtual storage at the printer.

The CMS commands described in this chapter are:

- SVCTRACE, which records information for all SVC calls. When the trace is terminated, the information recorded up to that point is printed at the system printer.
- SET AUTODUMP, which controls the creation of an automatic dump containing the DMSNUC area of CMS, storage management workarea, page allocation table, and the loader tables in the event of an abend. QUERY AUTODUMP returns the current setting of the SET AUTODUMP command.

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 of the information you need using the CP TRACE command. The SVCTRACE command is a CMS command, which provides more detailed information about all SVCs executed 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 *VM/SP CMS Command 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.

Since 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 executing. Use the Immediate command RO to resume tracing.

Since 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 execute in CMS, see *VM/SP Application Development Guide for CMS*.

Tracing Capabilities in EXECs

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

- System Product Interpreter
- EXEC 2
- CMS EXEC.

The amount of information displayed during execution of an EXEC is controlled by a single instruction. The instruction depends upon which processor is being used as shown below:

Processor	**Instruction**
System Product Interpreter	TRACE
EXEC 2	&TRACE
CMS EXEC	&CONTROL

Tracing can also be turned on for the System Product Interpreter or EXEC 2 by entering the following CMS command:

```
set exectrac on
```

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

The TRACE instruction used by the System Product Interpreter has several options to control how much information is displayed to the user. The TRACE instruction even allows you to enter interactive debug. During interactive debug, the interpreter pauses after almost every instruction allowing the user to single-step through the program.

Assume that we have a Restructured Extended Executor (REXX) program called STATUS EXEC, which gives us 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 execution
- Intermediate results during evaluation of expressions
- Substituted names.

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

```
User ID:  GEORGEB
Time   :  09:50:54
Date   :  Thursday, 7 Apr 1988
```

When the STATUS EXEC is executed *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, you either type:

```
trace off
```

to end debug or hit the **ENTER** key to continue executing 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, you either type:

```
trace off
```

to end debug or hit the **ENTER** key to continue executing 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 1988"
       >O>   "Date  :  Thursday, 7 Apr 1988"
Date  :  Thursday, 7 Apr 1988
```

At this point, you either type:

```
trace off
```

to end debug or hit the **ENTER** key to continue executing 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 three through six of STATUS EXEC were traced and execution stopped at each step.

The System Product Interpreter also has a TRACE function. See *VM/SP System Product Interpreter Reference* for more information on using the TRACE instruction and TRACE function.

Nucleus Load Map

Each time the CMS resident nucleus is loaded on a DASD and an IPL can be performed on that DASD, a nucleus load map is produced as a printer spool file. 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 *VM/SP Installation 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 exist in the loaded files. The load map is contained in the third record of the MODULE file. 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. Since 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.
- 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. Since MAP is the default option, just be sure that NOMAP is not specified. Issue the MODMAP command to type the load map associated with the specified MODULE file on the terminal. The format of the MODMAP command is:

MODmap	*filename*

filename
is the module whose map is to be displayed. The file type must be MODULE.

How to Print a CMS Dump File

Use the IPCSPRT command to print a previously created dump file, created using IPCSDUMP, under CMS. See the *VM/SP Interactive Problem Control System Guide and Reference* for more information.

Reading CMS Abend Dumps

If an abend dump is desired when CMS abnormally terminates, the terminal operator may enter:

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

By issuing these commands a dump spool file is created and sent to your reader. Now the system must be re-IPLed and then issue the IPCSDUMP command which will format the dump into a usable form. The dump formats and prints:

- 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 terminates. 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. Refer to *VM/SP Application Development Guide for CMS* for functional information on CMS. The following discussion on reading CMS dumps refers to several CMS control blocks and fields in the control blocks.

Refer to the *VM/SP CMS Data Areas and Control Blocks* for details on CMS control blocks. Figure 8 on page 137 shows the CMS control block relationships. You also need a current CMS nucleus load map to analyze the dump.

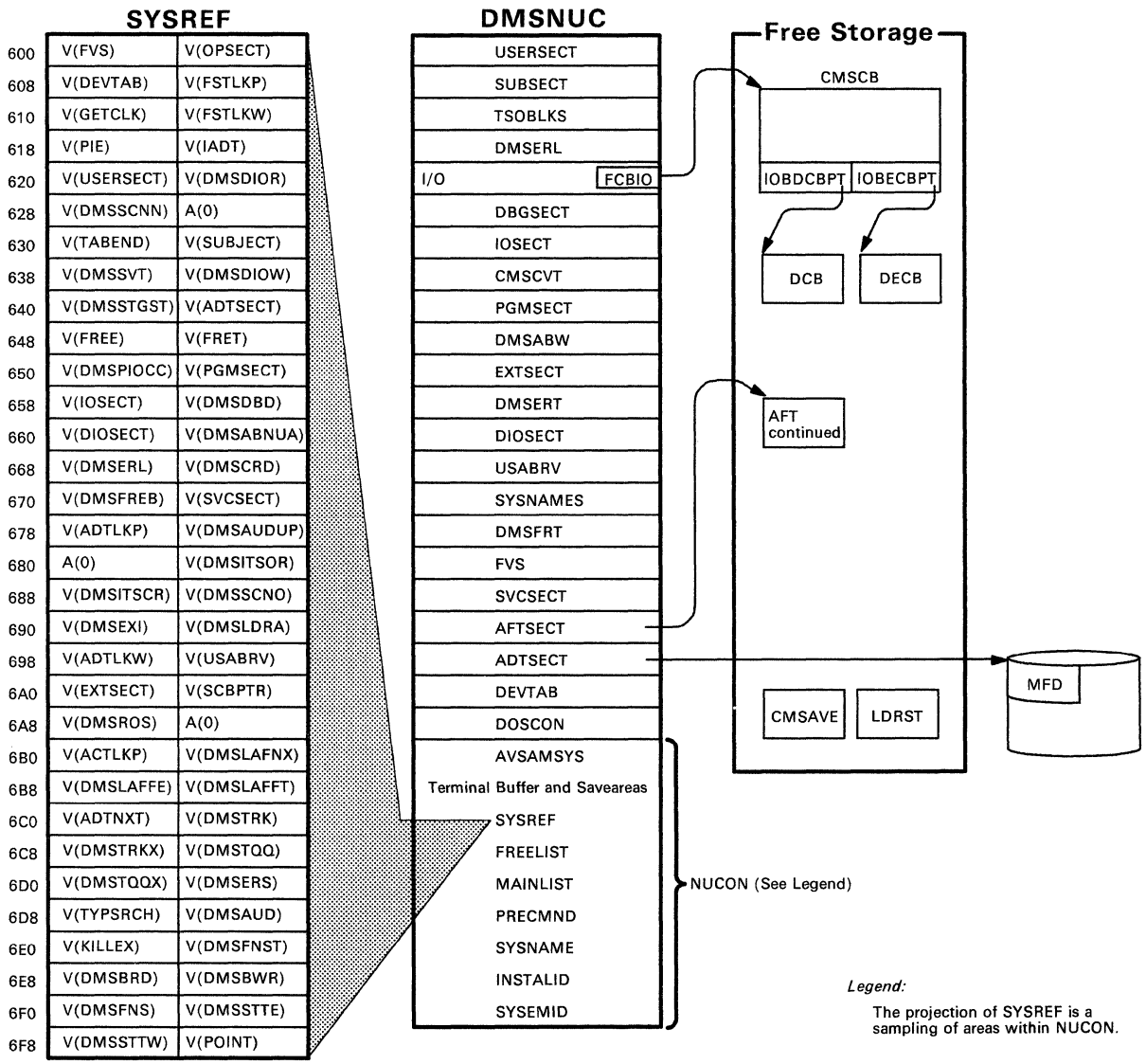


Figure 8. CMS Control Blocks

Generating CMS Abend Dumps Automatically

By using the SET AUTODUMP command, you can automatically generate a dump containing the DMSNUC area of CMS, storage management workarea, page allocation table, and loader tables if the system abends. SET AUTODUMP CMS dumps storage for the following system errors:

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

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
- Use of the ABEND macro
- Use of the DMSABN macro.

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

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

If you have set AUTODUMP to ALL or CMS, the dump containing the DMSNUC area of CMS, storage management workarea, page allocation table, and loader tables is produced using the CP VMDUMP facility. The reader of the virtual machine where the abend occurred, receives the dump. This user also receives a messages saying that the dump has been taken.

IPCSDUMP can process the dump using the IPCSSCAN CMSPOINT subcommand. For more information on IPCSDUMP and IPCSSCAN CMSPOINT subcommand, refer to the *VM/SP Interactive Problem Control System Guide and Reference*.

The SET AUTODUMP Command

Use the SET AUTODUMP command to generate a dump of the DMSNUC area of CMS, storage management workarea, page allocation table, and loader tables automatically if the system abends.

Format

SET	**AUTODUMP** [**CMS** **ALL** **OFF**]

Operands

CMS
dumps storage of the DMSNUC area, storage management workarea, page allocation table, and loader tables whenever an unrecoverable CMS system abend occurs and CMS enters a disabled wait state. CMS is the default setting.

ALL
dumps storage of the DMSNUC area, storage management workarea, page allocation table, and loader tables for all abends within the virtual machine, both recoverable and non-recoverable.

OFF
does not dump storage for any abends.

Note: A dump is not produced by the HX command. Also, if a program check is trapped using STAE, SPIE, or ABNEXIT, then the dump of DMSNUC is not produced.

The QUERY AUTODUMP Command

Use the QUERY AUTODUMP command to determine the current setting of the SET AUTODUMP command.

Format

QUERY	**AUTODUMP** [*(options...[)]*] *options:* [**STACK** [**FIFO**|**LIFO**] **FIFO** **LIFO**]

Options

STACK
causes the result of the QUERY command to be placed in the program stack instead of being displayed at the terminal. The information is stacked either FIFO (first-in, first-out) or LIFO (last-in, first-out). The default is FIFO.

If CMS passes the command to CP, then the response from CP is also put in the program stack. If CP precedes the QUERY command, CMS does not stack the results. The STACK option is valid only when issued from CMS.

FIFO
(first-in, first-out) is the default option for STACK. FIFO causes the results of the QUERY command to be placed in the program stack instead of being displayed at the terminal. The information is stacked FIFO. The options STACK, STACK FIFO, and FIFO are all equivalent.

LIFO
(last-in, first-out) causes the results of the QUERY command to be placed in the program stack instead of being displayed at the terminal. The information is stacked LIFO. This option is equivalent to STACK LIFO.

Response

- AUTODUMP = ALL
- AUTODUMP = CMS
- AUTODUMP = OFF

Reason for the Abend

Determine the immediate reason for the abend and identify the failing module. The abend message DMSABN148T contains an abend code and failing address. *VM/SP System 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 terminates.

You may have to examine several fields in the nucleus constant area (NUCON) of low storage.

1. Examine the program old PSW (PGMOPSW) at location X'28'. Using the PSW and current CMS nucleus load map, determine the failing address.
2. Examine the SVC old PSW (SVCOPSW) at location X'20'.
3. Examine the external old PSW (EXTOPSW) at location X'18'. If the virtual machine operator terminated CMS, this PSW points to the instruction executing when the termination request was recognized.
4. For a machine check, examine the machine check old PSW (MCKOPSW) at location X'30'. Refer to Figure 18 on page 251 for a description of the PSW.

Collect Information

Examine several other fields in NUCON to analyze the status of the CMS system. As you proceed with the 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 complete contents of NUCON and the other CMS control blocks are described in *VM/SP CMS Data Areas and Control Blocks*. The following areas of NUCON may contain useful debugging information.

- Save Area for Low Storage

  Before executing, DEBUG saves the first 160 bytes of low storage in a NUCON field called LOWSAVE. LOWSAVE begins at X'C0'.

- Register Save Area

  DMSABN, the abend routine, saves the user's floating-point and general purpose registers.

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

- Device

  The name of the device causing the last I/O interrupt is in the DEVICE field at X'26C'.

- Last Two Commands or Procedures Executed

Field	Location	Contents
LASTCMND	X'2A0'	Last command issued from the CMS or XEDIT command line. If a command issued in a CMS EXEC abnormally terminates, 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'	Next-to-last command issued from the CMS or XEDIT command line. If a command issued in a CMS EXEC abnormally terminates, 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'	Last EXEC procedure invoked. EXEC 2 and System Product Interpreter do not update this field.
PREVEXEC	X'2B8'	Next to last EXEC procedure invoked. EXEC 2 and System Product Interpreter do not update this field.

- Last Module Loaded into Free Storage and the Transient Area

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

- Pointer to CMSCB

  The pointer to the CMSCB is in the FCBTAB field located at X'5C0'. CMSCB contains the simulated OS control blocks. These simulated OS control blocks are 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 last command entered from the terminal is stored in an area called CMNDLINE (X'7A0'), and its corresponding PLIST is stored at CMNDLIST (X'848').

- External Interrupt Work Area

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

  - The PSW, EXTPSW.
  - Register save areas, EXSAVE1.
  - Separate area for timer interrupts, EXSAVE.

- 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.

- Program Check Interrupt Work Area

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

- 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.

- Simulated Communications Vector Table (CVT)

  The CVT, as supported by CMS, is CVTSECT. Only the fields supported by CMS are filled in.

- Active Disk Table and Active File Table

  For file system problems, examine the Active Disk Table (ADT), or Active File Table (AFT) in NUCON.

See a CMS nucleus map for the location of these CSECTs.

Register Use

To trace control blocks and modules, it is important to know the CMS GPR usage conventions.

Register	Contents
GPR1	Address of the PLIST
GPR12	Program's entry point
GPR13	Address of a 12-doubleword work area for an SVC call
GPR14	Return address
GPR15	Program entry point or the return code

The preceding information should help you to read a CMS dump. If it becomes necessary to trace file system control blocks, refer to Figure 8 on page 137 for more information. With a dump, the control block diagrams, and a CMS load map, you should be able to find the cause of the abend.

Tips for debugging after receiving a program check abend (e.g. DMSITP141) are as follows:

- DMSITP, the CMS program interrupt handler, 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 DMSABN 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 execution of a system routine, control is halted until the wait state bit is turned off or CMS is re-IPLed.

- 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 12 (X'C') bytes into the DSECT, immediately followed by registers 14, 15, 0, 1, and 2. The Program Interrupt Element (PIE), needed by SPIE, primarily uses these areas. Registers 0 through 15 are stored at location 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. Location 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 location 4 and the name of the routine being called is saved at location 8. At location X'10', the old PSW is stored, followed by the addresses for the normal return and the error return. The registers 0 through 15 are stored at location 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 terminated.

Commands that Alter the Contents of Storage

You can use the STORE and STCP commands to alter the contents of virtual machine storage and real storage.

ZAP and ZAPTEXT commands are used to alter modules, OS LOADLIBS, TEXT libraries, or TEXT decks before the code is loaded and executed.

ZAP and ZAPTEXT are described in the *VM/SP Installation Guide*. See "Altering the Contents of Virtual Machine Storage (STORE command)" on page 67 and "Altering the Contents of Real Storage (STCP command)" on page 70 for information on STORE and STCP.

Chapter 5. Debugging the SFS Server Machine

The three ways that you can collect error information for problem diagnosis are described in this chapter. They are:

- Using console logs, described in "Using the Console Log."
- Using dumps, described in "Using File Pool Server Dumps to Diagnose Problems" on page 150.
- Using system trace data, described in "Using System Trace Data to Diagnose Problems" on page 152.
- Using PER, described in "Using the CP PER Command" on page 59
- Using SVCTRACE, described in "Using the SVCTRACE command" on page 132.

Note: The SFS server operator does not necessarily diagnose problems, especially from the virtual machine. Dumps and system trace data are usually used by the system programmer or whoever is responsible for diagnosing system problems.

Summary of Steps to Follow When a File Pool Server Abend Occurs

When an SFS server abend occurs, you must do the following steps:

1. Collect information about the error.
   - Save the console sheet or spooled console output from the SFS server virtual machine.
   - Save and process any dumps that the SFS server produces.

     When an abend occurs in the SFS server, either because the SFS server issued an abend or because an SFS server or CMS operation caused a program exception, the SFS server produces a dump via the CP VMDUMP command (described in the *VM/SP CP General User Command Reference*). CP sends the dump to the SFS server's virtual reader.
   - Save any CPTRAP file that contains SFS server data.
2. Collect other types of information about system status, such as:
   - Status of real and virtual devices that the SFS server is using
   - System load at the time of the failure on any systems using the SFS server and the status of each system (for example, did another system abend?)
   - Types of applications that are using the SFS server at the time and any information about them
   - Physical connection configuration of the systems in use.

Using the Console Log

The SFS 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 SFS server virtual machine or another virtual machine user ID to whom you want the SFS server to send the console log. You

may want to add this to the SFS 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 *VM/SP CP General User Command Reference* for details on the SPOOL command.

The SFS 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:

- abend code
- program old PSW
- contents of the general purpose registers.

The SFS server also attempts to determine the displacement of the module in which the abend occurred and the displacement of the calling module.

Figures 9, 10, and 11 show some of the messages that the SFS server may issue in response to an abend condition:

```
DMSITP141T Operation exception occurred at 48C9FA in routine DMS5IF

SDS    ABEND SAVEAREA :

ADDR   OFFSET  DUMP DATA

53C0C4 000000  FFE000C1 4048C9FA 003E10C4 003E10A0  * ...A .I....D.... *
53C0D4 000010  00520C60 003E10C0 003E11D7 00000100  * ...-.......P.... *
53C0E4 000020  00537691 00000000 00000000 013E11FF  * ................ *
53C0F4 000030  00536DC0 003E1058 00509AB8 003E1058  * .._.......&..... *
53C104 000040  4050A002 0048C9F8                    *  &....I8        *

ABTERM CODE 0C1 AT 48C9F8

PROGRAM OLD PSW IS : FFE000C1 4048C9FA

GPR  0 =  003E10C4  003E10A0  00520C60  003E10C0
GPR  4 =  003E11D7  00000100  00537691  00000000
GPR  8 =  00000000  013E11FF  00536DC0  003E1058
GPR 12 =  00509AB8  003E1058  4050A002  0048C9F8

FAILURE   AT OFFSET +0479F8 IN DMSDAC  PROGRAM  (445000)
FAILURE   AT OFFSET -07D0C0 IN  DMS5BC    87.167

CALLED  FROM OFFSET +06AA22 IN DMSSAC  PROGRAM  (49F000)
CALLED  FROM OFFSET +000122 IN  DMS5BB    87.135

STORAGE NEAR FAILURE :

ADDR   OFFSET  DUMP DATA

48C9D8 000000  00CC0004 000000D0 00040000 00D40003  * .............M.. *
48C9E8 000010  00000000 000000C0 0049D640 0049D570  * ..........O ..N. *
48C9F8 000020  00000000 007210C4 D4E2F3E2 D4404040  * .......DMS3SM    *
48CA08 000030  40F8F74B F2F1F600 47F0F006 08BE90EC  *  87.216..00..... *
48CA18 000040  D00C05B0 41C0BFFF 5800C69D 5800B02C  * ..........F..... *

POTENTIAL WILD BRANCH AT : 50A000

BAL(R)    AT OFFSET +06B000 IN DMSSAC  PROGRAM  (49F000)
BAL(R)    AT OFFSET +000548 IN  DMS5BC    87.167

ADDR   OFFSET  DUMP DATA

509FE0 000000  58E0A010 58E0E014 50E0B06C 58A0A01C  * ........&..%.... *
509FF0 000010  4100B06C 5000B048 58F0E0D8 4110B048  * ...%&....0.Q.... *
50A000 000020  05EF58A0 A01C47F0 C5D2D507 2000C837  * .......0EKN...H. *
50A010 000030  4770C576 5020B048 41E0B068 50E0B04C  * ..E.&.......&..< *
50A020 000040  58F0C7BC 4110B048 05EF47F0 C5D25020  * .0G........0EK&. *



AB/00C1          PIDS/5749DMS00        RIDS/DMS5BC          ADRS/07D0C0
```

Figure 9. First Sample SFS Server Console Log

```
DMSITP141T Protection exception occurred at 4DF54E in routine DMS5IF

SDS    ABEND SAVEAREA :

ADDR   OFFSET  DUMP DATA

5140C4 000000  FFE000C4 904DF54E 0000015E 003D6204  * ...D.(5+...;.... *
5140D4 000010  80000000 003D6360 00000005 00000005  * .......-........ *
5140E4 000020  00000000 00000000 00000000 00000000  * ................ *
5140F4 000030  00273310 003D6300 004DF478 003D6300  * .........(4..... *
514104 000040  00000000 004DF478                    * .....(4.         *

ABTERM CODE 0C4 AT 4DF54A

PROGRAM OLD PSW IS : FFE000C4 904DF54E

GPR  0 =  0000015E  003D6204  80000000  003D6360
GPR  4 =  00000005  00000005  00000000  00000000
GPR  8 =  00000000  00000000  00273310  003D6300
GPR 12 =  004DF478  003D6300  00000000  004DF478

FAILURE   AT OFFSET +05054A IN DMSSAC  PROGRAM  (48F000)
FAILURE   AT OFFSET +0000D2 IN  DMS4ND    87.134

CALLED  FROM OFFSET +01B10E IN DMSSAC  PROGRAM  (48F000)
CALLED  FROM OFFSET +00042E IN  DMS4DK    87.195

STORAGE NEAR FAILURE :

ADDR   OFFSET  DUMP DATA

4DF528 000000  B0604740 C08A58E0 300054E0 C1A412EE  * .-. ........A... *
4DF538 000010  4770C10C 47F0C102 58403004 D207B060  * ..A..0A.. ..K..- *
4DF548 000020  40009203 40081F55 BF534009 5050B054  *  ... ..... .&&.. *
4DF558 000030  41E0B054 50E0B04C 58F0C1AC 4110B04C  * ....&..<.0A....< *
4DF568 000040  05EF12FF 4780C102 58F0C1B0 4110C1A0  * ......A..0A...A. *



SAC termination during forward processing
   LUWID = C3A                   USERID = LUDWIG3
   OPERATION = INSERT
   CATALOG-ID = 6505             INDEX-ID = 6512
   PAGE-ADDRESS = 2C8000         PAGE-TYPE = INDEX
   PAGE-NUMBER = DE2

AB/00C4          PIDS/5749DMS00        RIDS/DMS4ND          ADRS/0000D2
```

Figure 10. Second Sample SFS Server Console Log

```
File pool server system error occurred - DMS4NA 13

SDS    ABEND SAVEAREA :

ADDR   OFFSET  DUMP DATA

57D0C4 000000  00000000 4054AF0C 00000000 0054AFD0  * .... ..........  *
57D0D4 000010  00000004 00000000 0041C870 003AD0C0  * ..........H.....  *
57D0E4 000020  00000005 000000C4 0041C240 003AD060  * .......D..B ...-  *
57D0F4 000030  003FF318 003FDF08 4054A3DA 003FDF08  * ..3..... .......  *
57D104 000040  4054AF0E 0056D508                    *  .....N.          *

GPR  0 =  00000000  0054AFD0  00000004  00000000
GPR  4 =  0041C870  003AD0C0  00000005  000000C4
GPR  8 =  0041C240  003AD060  003FF318  003FDF08
GPR 12 =  4054A3DA  003FDF08  4054AF0E  0056D508

FAILURE   AT OFFSET +04CF0A IN DMSSAC  PROGRAM  (4FE000)

CALLED  FROM OFFSET +04DF3C IN DMSSAC  PROGRAM  (4FE000)
CALLED  FROM OFFSET +00029C IN  DMS4NH    87.244

SAC termination during forward processing
   LUWID = 15E0                  USERID = LUDWIG3
   OPERATION = BULK INSERT
   CATALOG-ID = 6503
   PAGE-ADDRESS = 428000         PAGE-TYPE = INDEX
   PAGE-NUMBER = E20

MS/DMS3040E       PIDS/5749DMS00        RIDS/DMS4NA          PRCS/13
```

Figure 11. Third Sample SFS Server Console Log

Using File Pool Server Dumps to Diagnose Problems

You can use IPCS to collect and diagnose problem data for the SFS server virtual machine. The console listing, as described in "Using the Console Log" on page 146, may help you diagnose problems without using dumps.

The steps involved in using dumps to diagnose problems are:

1. Create the SFS file pool server dump
2. Process the SFS server dump
3. Diagnose the SFS server dump
4. Print the SFS server dump.

Creating an SFS File Pool Server Dump

The SFS server virtual machine creates its own dumps. The dump goes to the reader of the SFS server virtual machine. Because the SFS server virtual machine is not set up to process dumps, you need to transfer the dump file to the appropriate virtual machine.

If the SFS server virtual machine cannot create the dump, you can use the VMDUMP command. The VMDUMP command dumps virtual storage that VM/SP creates for the virtual machine user; in this case, for the SFS server. The dump goes to the virtual machine specified by the SYSDUMP parameter on the SYSOPR macro in the DMKSYS ASSEMBLE file, if you enter the following CP command:

```
vmdump 0-end system format sfs
```

Do not use the reserved names of ATSCAB1 or ATSCAB2 for the dump ID of VMDUMP. See the *VM/SP CP General User Command Reference* for more information on the VMDUMP command.

Processing an SFS File Pool Server Dump

After the SFS server virtual machine creates a dump, load the dump onto disk. To load the dump, enter the following IPCS command:

```
ipcsdump
```

The default map file is SFSIPCS MAP.

When you issue IPCSSCAN, it invokes an SFS server routine to extract information from the dump and transmit it to IPCS for inclusion in the problem report and/or symptom summary. IPCSDUMP creates the following:

- Problem report
- Symptom summary
- Disk resident dump to which IPCS appends the map information.

See the *VM/SP Interactive Problem Control System Guide and Reference* for more information about the IPCSDUMP command.

Diagnosing an SFS File Pool Server Dump

The IPCSDUMP command generates a symptom record, which is based on problem report information. The symptom record helps you find out why the SFS 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:
  - Error code
  - ID of the failing component
  - ID of the failing module
  - Registers and PSW contents.

You can also use the IPCSSCAN command to examine the dump interactively. The IPCSSCAN command is described in *VM/SP Interactive Problem Control System Guide and Reference*. The following sections introduce those subcommands specifically for the SFS server.

Note: TRACE can also be used for CP dumps.

Formatting and Displaying Trace Records

You can scroll through the formatted output with either of the following IPCS IPCSSCAN subcommands:

- TRACE SCROLL or SCROLLU
- SCROLL or SCROLLU.

See *VM/SP Interactive Problem Control System Guide and Reference* for more information about the IPCSSCAN TRACE and SCROLL subcommands.

Printing a File Pool Server Dump

The IPCSPRT command prints the dump and symptom record that IPCSDUMP processed. The output you get consists of the following:

- Symptom record
- Dump in hexadecimal (no special formatting)
- Contents of the registers and the PSW.

See *VM/SP Interactive Problem Control System Guide and Reference* for more information on the IPCSPRT command.

Using System Trace Data to Diagnose Problems

The SFS server maintains an internal trace table within the SFS server virtual machine. You can use the IPCS IPCSSCAN TRACE subcommand to display the internal trace table entries. The SFS server also writes trace entries to the system CPTRAP file. You can then use IPCSSCAN to view SFS server entries.

External Tracing

The SFS server ETRACE command lets you enable or disable external tracing for the SFS server virtual machine. If you want to collect SFS server trace records, issue the following from the SFS server virtual machine after CPTRAP is started:

```
etrace on
```

After you issue ETRACE ON a series of prompts will 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. A single user ID or all user IDs (*) can be specified.
- for what type of SFS server processing will be traced. DAC and/or SAC can be specified as types of SFS server processing.
- for SFS server tracing of the subcomponents and the trace level desired.

A 0 entered for a prompt will cancel the ETRACE command.

If you want to stop tracing for the SFS server machine enter:

```
etrace off
```

You may also start tracing, using ETRACE, by specifying the proper start-up parameters when the SFS server machine is started.

To process the trace output use IPCS to view the results.

When you set external tracing on, certain internal SFS server trace records are written externally to a CPTRAP spool file. A complete description of the ETRACE command is in the *VM/SP CMS Shared File System Administration.*

Internal Tracing

The SFS server ITRACE command lets you enable or disable internal tracing for the SFS server virtual machine. If you want to collect SFS server trace records, issue the following from the SFS server virtual machine after CPTRAP is started:

```
itrace on
```

If you want to stop tracing for the SFS server machine enter:

```
itrace off
```

ITRACE is used to trace APPC/VM communications between the SFS server machine and CMS users.

You may also start tracing, using ITRACE, by specifying the proper start-up parameters when the SFS server machine is started.

To process the trace output use IPCS to view the results.

A complete description of the ITRACE command is in the *VM/SP CMS Shared File System Administration.*

Using CPTRAP to Trap Trace Table Entries

The CPTRAP command collects SFS server information in a reader file. This information helps with problem determination.

Note: Because the SFS server virtual machine is not set up to diagnose problems, the virtual machine that has the authority to issue the CPTRAP command must do so.

The following commands activate CPTRAP for SFS server records only:

`cptrap id` *trapid* `ype gt allowid` *userid* `3e`

userid is the SFS server virtual machine user ID.
This activates CPTRAP for 3E entries that the SFS server virtual machine produces.

`cptrap enable id` *trapid*

Enter:

```
cptrap stop close
```

to end CPTRAP processing. When you issue this command, the CPTRAP SPOOL file goes to your reader.

For more specific information about the CPTRAP command, see "Debugging with the CPTRAP Facility" on page 95 and the *VM/SP CP System Command Reference.*

Viewing CPTRAP Data with IPCSSCAN

To access the CPTRAP reader file and review the entries contained in that file, enter the following:

```
ipcsscan trpnnnnn
```

where TRP*nnnnn* is the number of the CPTRAP reader file. The file name and file type are "TRP*nnnnn* CPTRAP."

Then when IPCSSCAN prompts for selectivity, enter:

```
3e machtype sds
```

This specifies you want to collect specific machine type entries for SFS server of record type 3E.

For more specific information about the IPCSSCAN command, see the *VM/SP Interactive Problem Control System Guide and Reference*.

Chapter 6. Debugging GCS

While running programs on the Group Control System (GCS), you can encounter the following types of problems:

- Loops
- Abends
- Incorrect results
- Endless wait states.[5]

To help you deal with these problems, GCS provides:

- Internal tracing facilities (See page 157.)
- External tracing facilities (See page 173.)
- Dumping facilities (See page 179.)
- Interactive Debugging Support (See page 181.).

Internal Tracing Facilities

In common storage, the GCS supervisor maintains a wraparound[6] trace table that serves all virtual machines in a group. When building your GCS configuration file, you specify how big you want this table to be. The minimum you can choose is 4K; the maximum depends upon how much common storage you have available to use. If you don't set a size limit, GCS gives you a default size of 16K. See the *VM/SP Installation Guide* for more information about defining a GCS group configuration file.

This table contains information about the following supervisor events:

- Task dispatches
- External interrupts
- I/O interrupts
- Program interrupts
- SVC interrupts
- I/O requests (SIO, DIAGNOSE, HDV, TIO) (invoked by supervisor)
- APPC/VM Synchronous event
- IUCV Signal System Service detail entries
- SVC GETMAIN storage requests
- SVC FREEMAIN storage requests
- Branch Entry FREEMAIN storage requests
- Branch Entry GETMAIN storage requests.

Besides tracing supervisor events, this table can record data from any of your GCS programs. The internal tracing of supervisor events is activated as soon as your virtual machine joins a group. Activating internal tracing of program data (GTRACE events) in your virtual machine involves the following: issuing the ITRACE command and then issuing the GTRACE macro.

[5] Outlined in Chapter 1, "Introduction to Debugging" on page 1.

[6] "Wraparound" means that, when the table fills, it goes back to the top and starts writing over itself.

Using the ITRACE Command and GTRACE Macro

To begin tracing data in a virtual machine, you must issue, 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 issue the ITRACE command.).

You can issue the ITRACE command for:

- Individual virtual machines, or
- Entire virtual machine group.

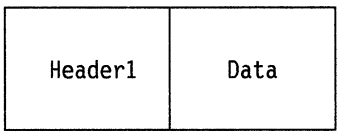
But any virtual machine operator who issues it on behalf of the whole group (ITRACE GROUP) must have an authorized user ID.

For more information about the ITRACE command and the GTRACE macro, see the *VM/SP Group Control System Command and Macro Reference*.

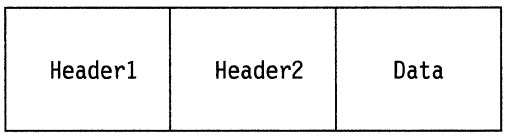
Formats of Internal Trace Entries

Entries in the internal trace table come in two different formats:

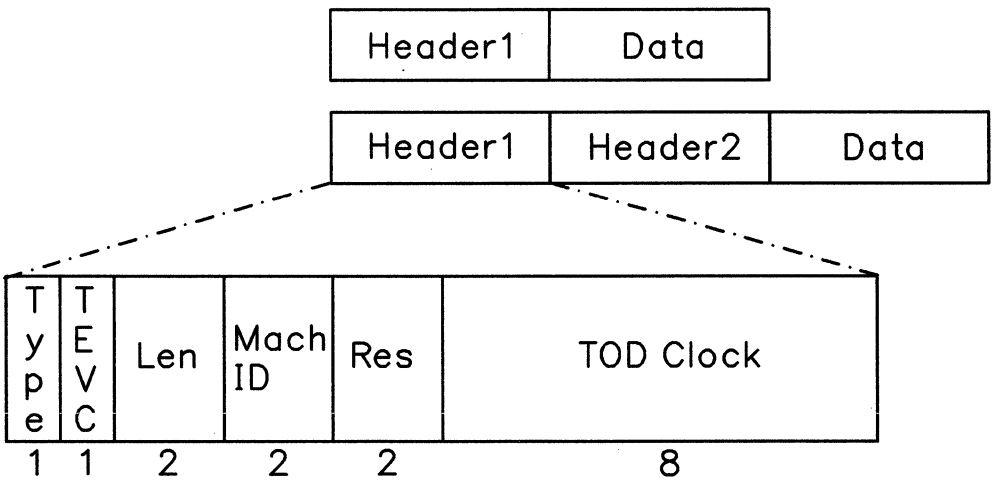
Supervisor entries:



GTRACE entries:



The Header1 that both entries share looks like this:



Type
Shows the type of trace entry:

X'01' = Dispatcher
X'02' = External Interrupt
X'03' = I/O Interrupt
X'04' = Program Interrupt
X'05' = SVC Interrupt
X'06' = I/O Request
X'07' = IUCV Signal System Service details
X'08' = SVC GETMAIN Request
X'09' = SVC FREEMAIN Request
X'0A' = GETMAIN Request via branch entry
X'0B' = FREEMAIN Request via branch entry
X'0C' = APPC/VM Synchronous event entry
X'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 zero (X'00'). Each time the table wraps, this number increases by one until it reaches X'FF'. After that, it recycles to X'00'.

By looking at this number, you'll be able to identify entries left over from the last "wrap" of the table. This could be important, for example, in a case where 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.

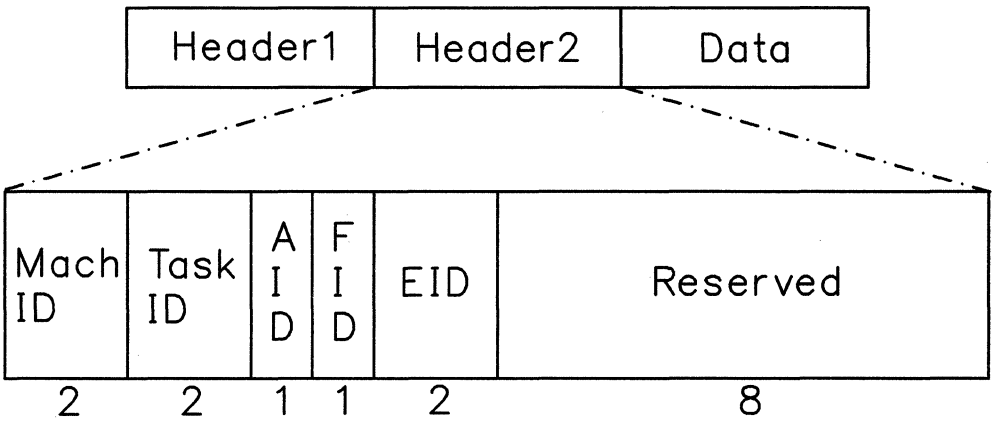
Len
Contains the length of the whole entry, including this header.

Mach ID
Identifies the virtual machine associated with this entry. There is a single trace table for the entire GCS group, it is important that you have the proper virtual machine identification (Mach ID).

Res
Represents a two-byte reserved field.

TOD Clock
Shows what time this entry was created in time-of-day format.

The Header2 used with GTRACE entries looks like this:



Mach ID
Identifies the virtual machine associated with this entry. There is a single trace table for the entire GCS group, it is important that you have the proper virtual machine identification (Mach ID).

Task ID
Identifies the task being traced.

AID
Indicates this is a "data" record. It always contains X'FF'.

FID
(Format ID) identifies what formatting module handles this entry.

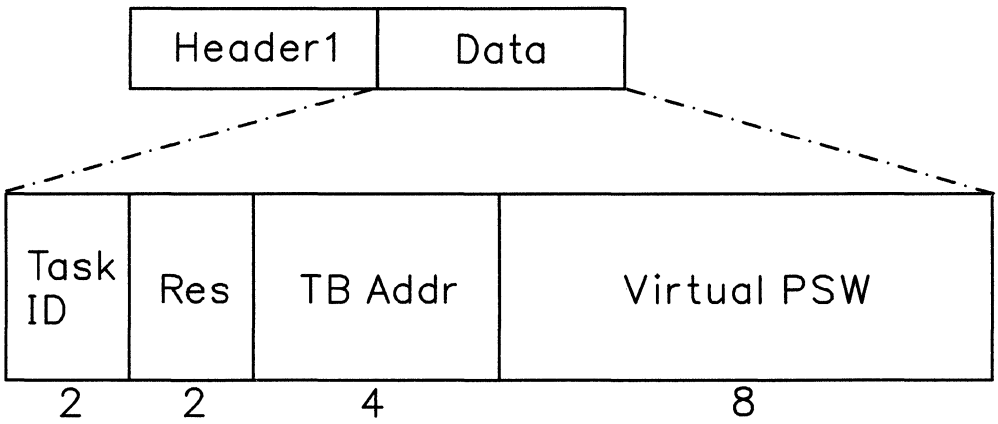
EID
Contains information from the GTRACE macro's ID parameter.

Reserved
Represents an eight-byte reserved field.

The data portion of supervisor entries can have any of twelve different formats:

- Dispatcher (type X'01'), see page 161
- External Interrupt (type X'02'), see page 162
- I/O Interrupt (type X'03'), see page 163
- Program Interrupt (type X'04'), see page 163
- SVC Interrupt (type X'05'), see page 164
- SIO (type X'06'), see page 165
- IUCV Signal System Service (type X'07'), see page 166
- Getmain via SVC (type X'08'), see page 167
- Freemain via SVC (type X'09'), see page 168
- Branch Entry Getmain (type X'0A'), see page 169
- Branch Entry Freemain (type X'0B'), see page 170
- APPC/VM Synchronous Event (type X'0C'), see page 171.

• **Dispatcher** (type X'01')



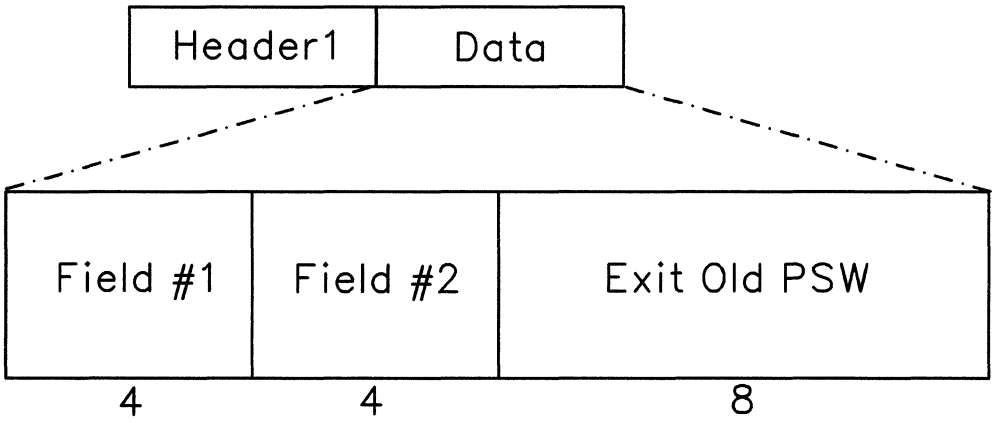
Task ID
Identifies the task being traced.

Res
Represents a reserved field.

TB Addr
Holds 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')



Field #1
The value of this field depends on the type of external interrupt. The interrupt code is stored in bytes 2 and 3 of the EXT old PSW.

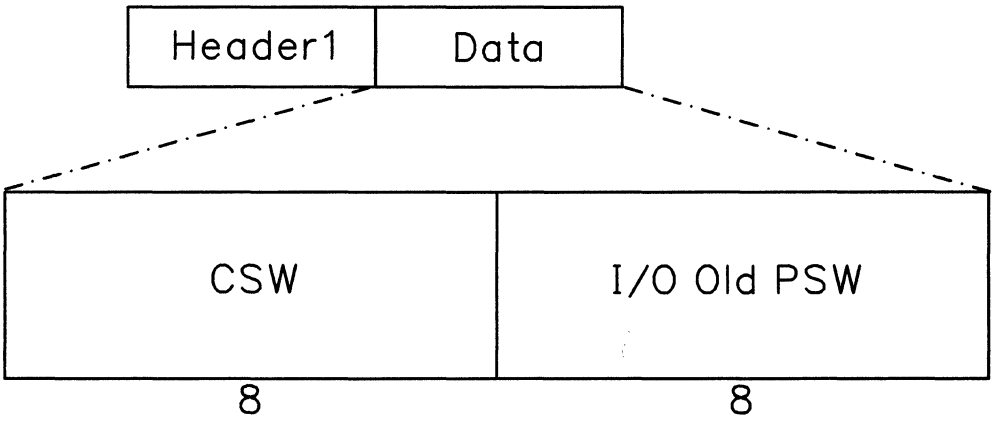
- For a Timer interrupt (code X'1004') this is a reserved field.
- For an IUCV interrupt (code X'4000') it contains:
  - 2-byte IPPATHID
  - 1-byte IPFLAGS1
  - 1-byte IPTYPE.
- For all other types of external interrupts this is a reserved field.

Field #2
The value of this field depends on the type of external interrupt. The interrupt code is stored in bytes 2 and 3 of the EXT old PSW.

- For a Timer interrupt (code X'1004') it contains a pointer to the Timer Queue Element.
- 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:
  - 2-byte IPCODE.
  - 1-byte IPWHATRC - for a connect pending (type X'81') interrupt, this byte will contain the IPFLAGS2 field.
  - 1-byte IPSENDOP.
- For all other types of external interrupts this is a reserved field.

Ext Old PSW
Contains the external old PSW. If an IUCV poll (rather than an external interrupt) generates this entry, the external old PSW will contain zeros (except for the interrupt code).

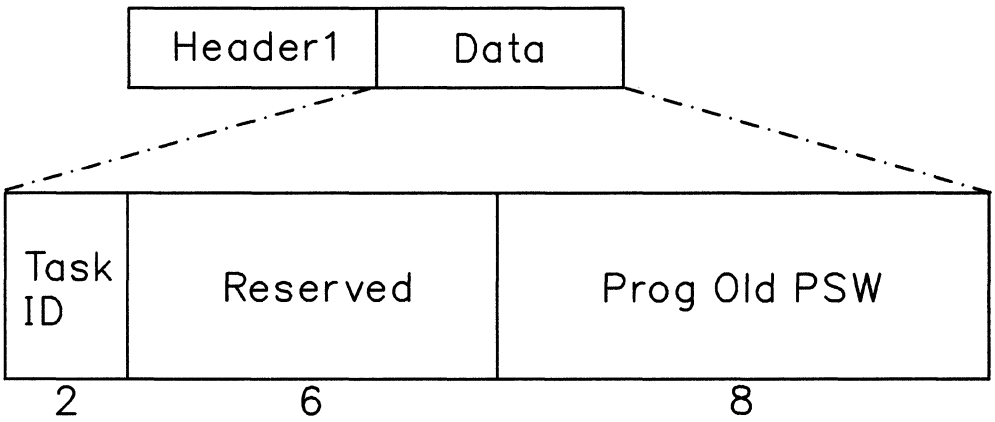
- **I/O Interrupt** (type X'03')



CSW
Contains the channel status word.

I/O Old PSW
Contains the old I/O PSW.

- **Program Interrupt** (type X'04')

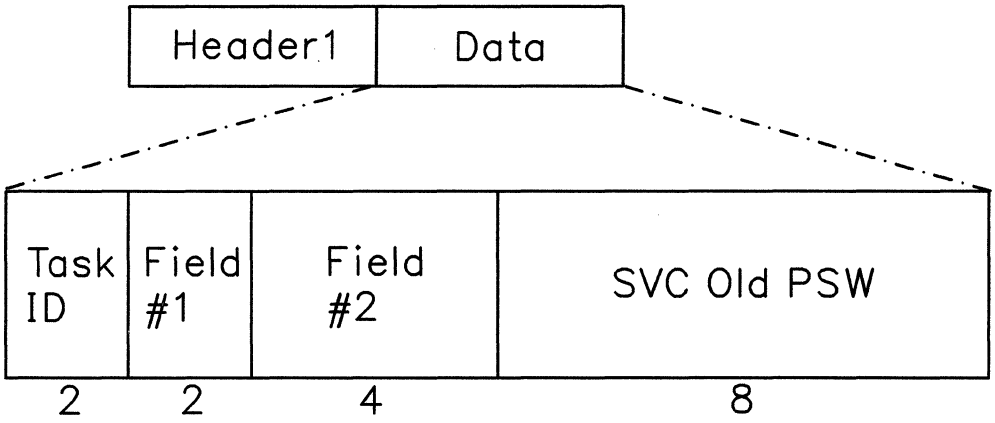


Task ID
Identifies the task being traced.

Reserved
Represents a reserved field.

Prog Old PSW
Contains the program old PSW.

• **SVC Interrupt** (type X'05')



Task ID
Identifies the task being traced.

Field #1
Represents a reserved field for all but three SVCs.

For SVC 202, it contains the first two bytes of the requested function's name. (For example, "GE" for the GENIO function.)

For SVC 203, it contains a two-byte flag and code parameter.

For a DOS SVC, the leftmost bit of this field is set to one, and the rest of the two bytes is zeros (reserved).

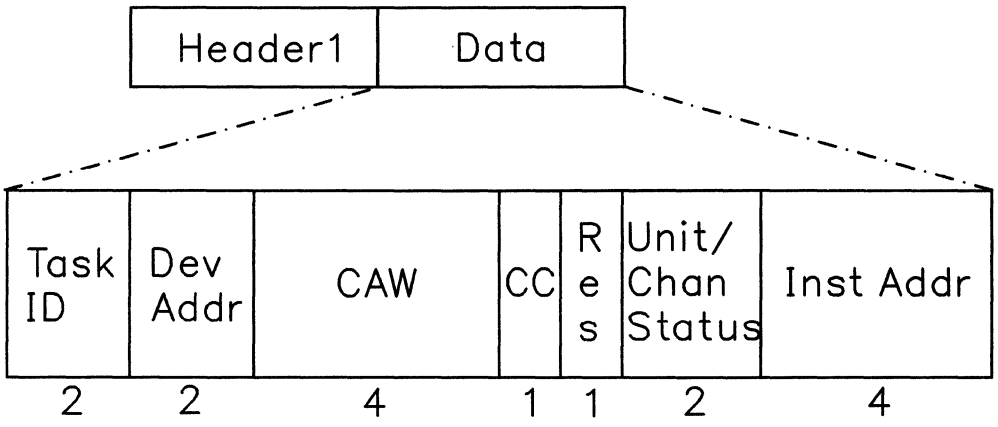
Field #2
Shows the contents of Register 1 for all SVCs except 202.

For SVC 202, it contains the middle four bytes of the requested function's name.

SVC Old PSW
Contains the entire eight bytes of the SVC old PSW for all SVCs but 202.

For SVC 202, it contains the last two bytes of the function name that's being invoked, followed by the last six bytes of the SVC old PSW.

•**SIO** (type X'06')



Task ID
Identifies the task being traced.

Dev Addr
Holds the virtual address of a device where the operation is being directed. (For a Test Channel instruction, this is the virtual channel address.)

CAW
Contains the channel address word. However, for instructions like HDV and TCH that don't use CAWs, this contains zeros.

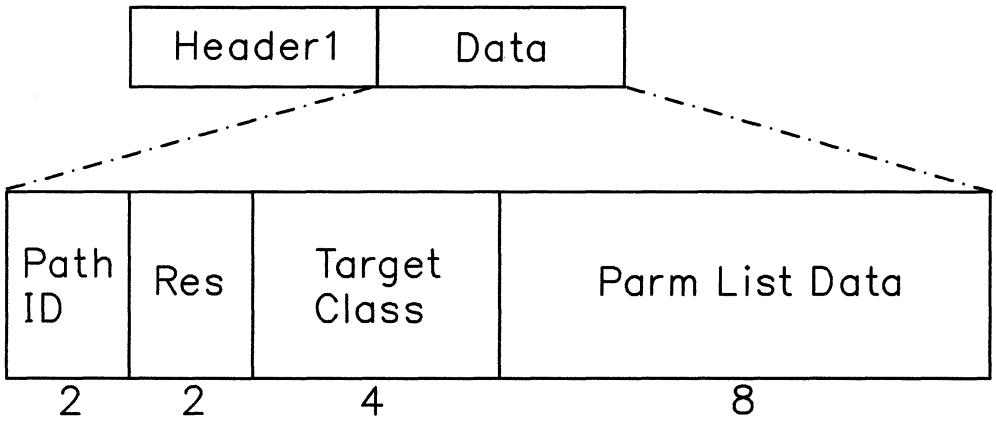
CC
Contains the condition code from the SIO event. The field is relevant only for SIOF and DIAGNOSE code X'98' SIO entries.

Res
Represents a reserved field.

Unit/Chan Status
Contains the unit and channel status bytes from the stored CSW. This field is relevant only for SIOF and DIAGNOSE code X'98' SIO events, and only when the CC field = X'01'.

Inst Addr
contains the address of the I/O instruction (SIO, SIOF, TIO, HIO, HDV, CLRIO, TCH, DIAGNOSE code X'18', DIAGNOSE code X'20', or DIAGNOSE code X'98').

• **IUCV Signal System Service** (type X'07')



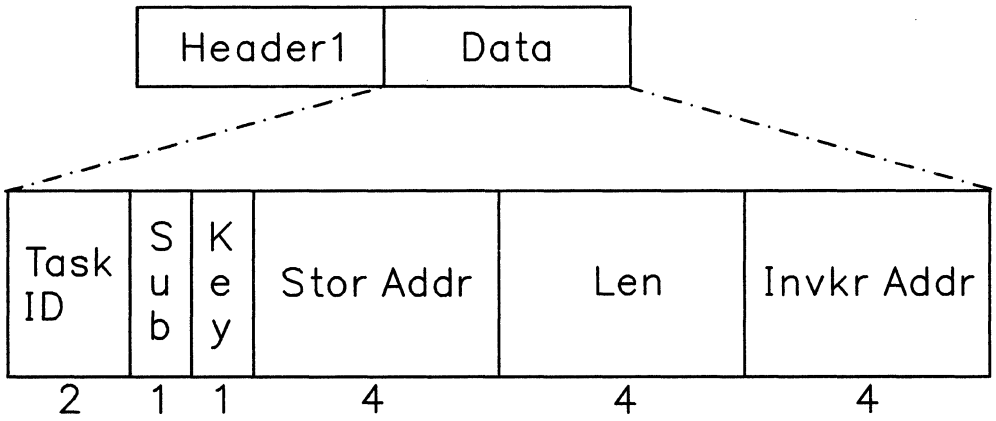
Path ID
Identifies a two-byte IUCV path.

Res
Represents a reserved field.

Target Class
Identifies an IUCV target class containing the interrupt source's Signal ID and type of signal sent.

Parm 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 invalid mode byte.

Key
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 invalid mode byte.
- If either the length or the subpool is incorrect, or both.

Note: The key data in this byte is right-justified. 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 will be 0 (zero).

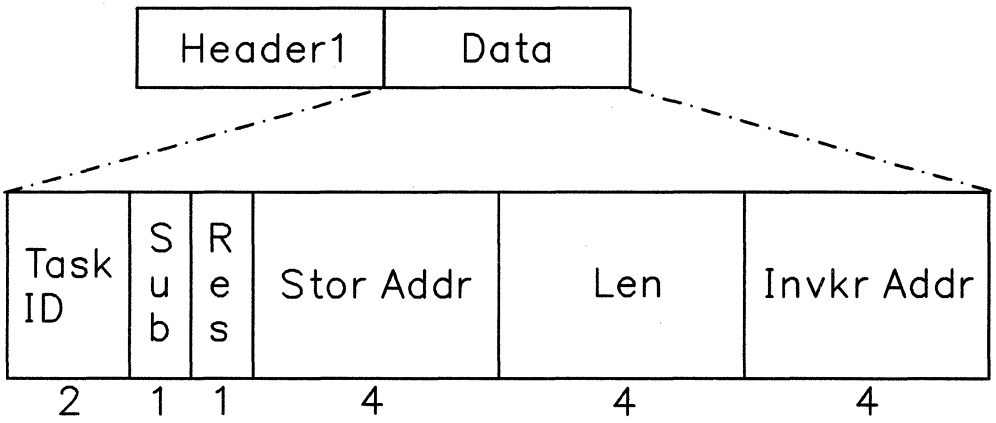
Stor Addr
Contains the address of storage obtained. If the GETMAIN failed, it contains zeros.

Len
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 invalid mode byte.

Invkr Addr
Identifies the invoker's address (the address that follows the 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 for the following failures:

- SVC 5 is issued with an invalid parameter list address.
- An unsupported MVS parameter is specified on the FREEMAIN macro.
- An invalid mode byte is encountered.

Res
Represents a reserved field.

Stor Addr
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:

- SVC 5 is issued with an invalid parameter list address.
- An unsupported MVS parameter is specified on the FREEMAIN macro.
- An invalid mode byte is encountered.

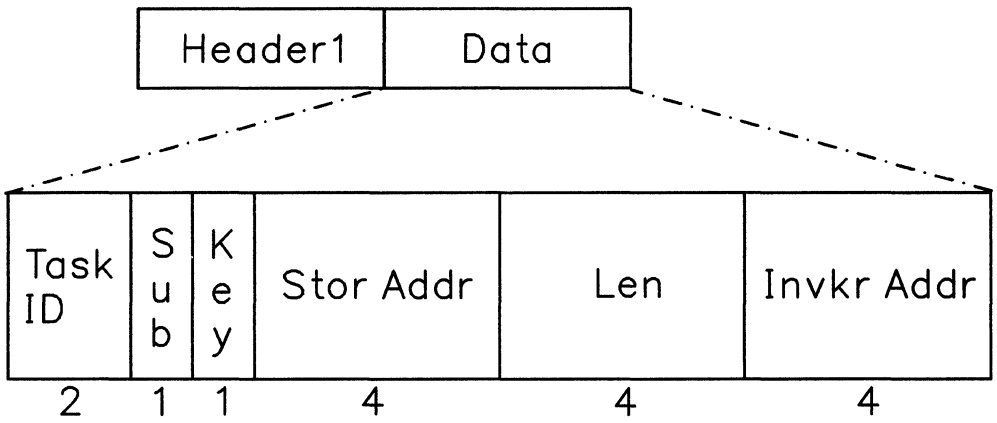
Len
Contains the length of the storage released. If the FREEMAIN fails, it contains the length passed to FREEMAIN.

It contains zeros for the following failures:

- SVC 5 is issued with an invalid parameter list address.
- An unsupported MVS parameter is specified on the FREEMAIN macro.
- An invalid mode byte is encountered.

Invkr Addr
Identifies the invoker's address (the address that follows the SVC).

- **Branch Entry Getmain** (type X'0A')



Task ID
Identifies the task being traced.

Sub
Contains the subpool specified in the GETMAIN request.

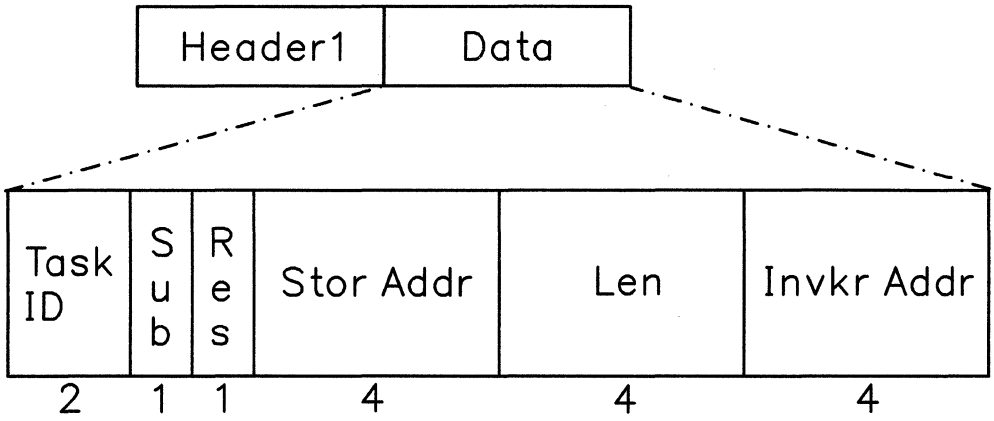
Key
The key data for a branch entry will always be filled in.

Stor Addr
Contains the address of storage obtained. If the GETMAIN failed, it contains zeros.

Len
Contains the length of the storage requested.

Invkr Addr
identifies the address following the GETMAIN call.

• **Branch Entry Freemain** (type X'0B')



Task ID
Identifies the task being traced.

Sub
Contains the subpool specified in the FREEMAIN request.

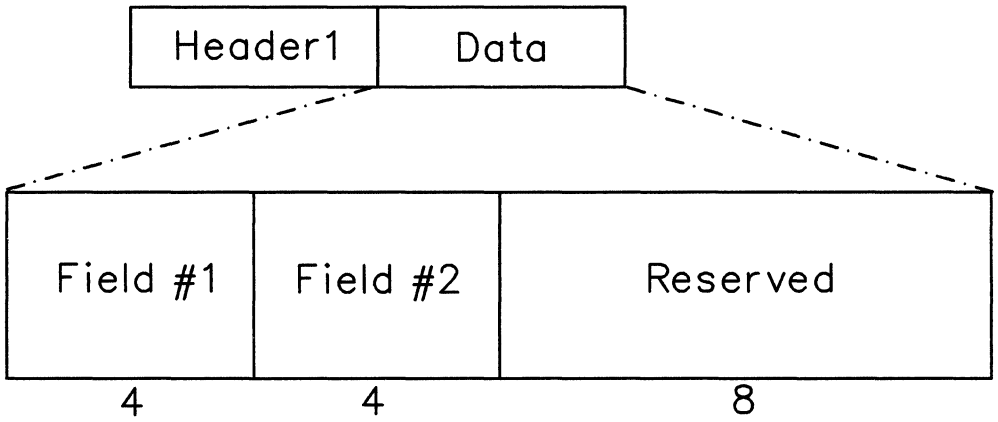
Res
Represents a reserved field.

Stor Addr
Contains the address of storage being released. If the FREEMAIN fails, it contains the storage address passed to FREEMAIN.

Len
Contains the length of the storage released. If the FREEMAIN fails, it contains the length passed to FREEMAIN.

Invkr Addr
identifies the address following the FREEMAIN call.

- **APPC/VM Synchronous Event** (type X'0C')



Field #1
contains:

- 2-byte IPPATHID
- 1-byte IPFLAGS1
- 1-byte IPTYPE.

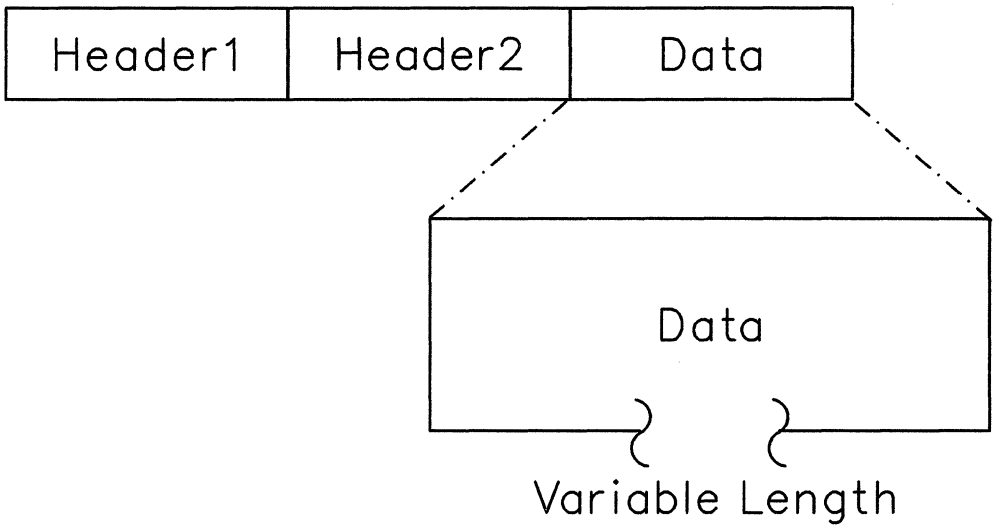
Field #2
contains:

- 2-byte IPCODE.
- 1-byte IPWHATRC - for a connect pending (type X'81') interrupt, this byte will contain the IPFLAGS2 field.
- 1-byte IPSENDOP.

Reserved
is a reserved field.

The data portion of the GTRACE entries looks like this:

- **GTRACE** (type X'0E')



Data
Represents from 1 up to 256 bytes of data passed from the application by the GTRACE macro.

How to Look at Internal Trace Table Entries

FLSCTB contains the common trace block address. The common trace block contains pointers in the following order:

1. Beginning of the table
2. End of the table
3. Next available entry slot in the table.

The next available entry immediately follows the last entry written. With that information, you can display whatever portion of the internal trace table that you need. For more information on FLSCTB refer to the *VM/SP Group Control System Command and Macro Reference*.

External Tracing Facilities

You can collect trace data in the CPTRAP spool file for later formatting and viewing. This requires a two-step process:

1. Issuing CPTRAP commands
2. Issuing the ETRACE command.

See the *VM/SP CP System Command Reference* and "Debugging with the CPTRAP Facility" on page 95 for more information on the CPTRAP command. See the *VM/SP Group Control System Command and Macro Reference* for more information on the ETRACE command. You must issue the following CPTRAP commands:

1. **CPTRAP ID** *gcs1* **TYPE GT GROUPID** *groupname* **3D**

   The GROUPID operand tells CPTRAP which virtual machine group to get the entries from. The X'3D' operand tells CPTRAP that it will be collecting "group" entries.

   Instead of GROUPID, you could use the ALLOWID operand to identify which particular virtual machine to get the entries from.

2. **CPTRAP ENABLE ID** *gcs1*

   The ENABLE operand starts the CPTRAP facility to start recording trace entries in the CPTRAP spool file.[7] But before GCS passes any records to CPTRAP, you must enable external tracing with the ETRACE command.

The virtual machine that issues the CPTRAP command must have a privilege class C user ID defined in the user ID's VM/SP directory entry. Once the CPTRAP commands have been issued, an authorized virtual machine in the group can issue the ETRACE command to start tracing for its own application or ETRACE GROUP command for tracing an entire group. ETRACE allows you to specify which of the following events should be traced and recorded in the CPTRAP spool file:

- Task dispatches
- External interrupts
- I/O interrupts
- Program interrupts
- SVC interrupts

[7] This spool file can be a wraparound file.

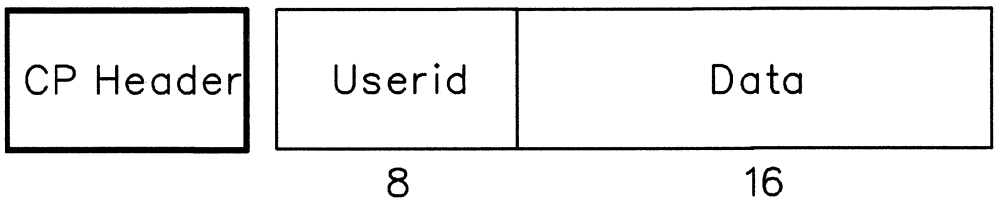
- I/O requests (SIO and DIAGNOSE)
- IUCV Signal System Service details
- APPC/VM synchronous events
- GETMAIN requests
- FREEMAIN requests
- User trace data generated via GTRACE macro.

Note: Since CPTRAP is a CP function, data in this external trace file may be mixed with trace data from CP, from other GCS machines or machine groups, and from machines not part of any GCS group. You must be selective when deciding what machines and what events to trace with CPTRAP. If you trace too many events, CPTRAP might create spool data faster than you can write it to a DASD, and you might lose data.

Formatting and Displaying External Trace Records

The external trace file contains two different entries produced by GCS virtual machines:

- An entry for GCS supervisor records:



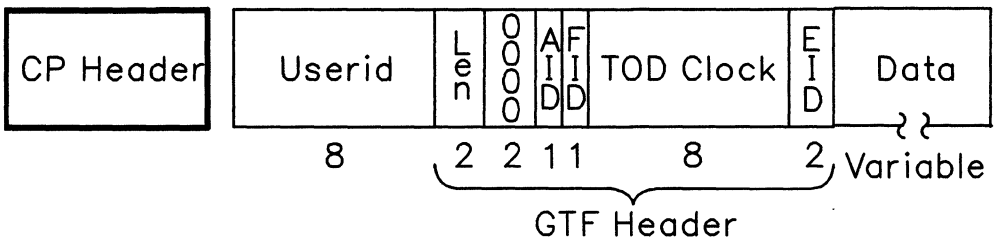
CP header
Contains a variable-length header appended by CPTRAP when it gets the record.

Userid
Identifies the virtual machine that the entry belongs to.

Data
Contains the data portion of the event's internal trace entry.[8]

[8] Internal trace entry formats begin on page "Formats of Internal Trace Entries" on page 158.

• An entry for GTRACE records:



CP header
Contains a variable-length header appended by CPTRAP when it gets the record.

Userid
Identifies the virtual machine that the entry belongs to.

Len
Contains the length of the entry, including the GTF header.

0000
Reserved field of the GTF header.

AID
Always contains X'FF', indicating that this is a data record.

FID
(Format ID) identifies the formatting module used for this record.

TOD Clock
Tells 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 256 bytes).

Primarily, you create an external spool file with CPTRAP to print out or interactively display your trace information. IPCS provides the commands to let you do this.

When IPCS selects a X'3D' (GCS) entry for displaying, it gives control to the GCS trace formatting routines. The format routines produce the output lines, but IPCS actually sends the lines to either the terminal or the printer. The format routines format the supervisor records and GTF header information. However, the applications being traced via the GTRACE facility have to supply their own GTRACE formatting modules. If they don't, their trace entries for the data portions of the records simply get printed unformatted, in hexadecimal.

The formatting routines handle formatting of supervisor entries differently from GTRACE entries. IPCS invokes a routine called CSIYTD. For supervisor records, CSIYTD will call a GCS-supplied formatting routine named CSIYTS. For supervisor records, CSIYTD will call a GCS-supplied formatting routine named CSIYTS to format it. However, for GTRACE records, CSIYTD uses GCS-supplied formatting routines to format the GTF header part of the record. CSIYTD also will look for another formatting routine, one supplied by the traced application, to finish the data portion of the record. (It uses the GTRACE record's one-byte FID field to

locate this routine. The routine's name must be CSIYTXxx, with "xx" being the two-digit FID, and it must have a file type of TEXT.)

If the CSIYTD program cannot find a user-supplied formatting routine, it prints the entry information in hexadecimal. If the program does find a CSIYTXxx TEXT, it calls that routine. At that time, the registers will contain:

R15 The CSIYTXxx routine's entry point

R14 The return address

R13 A 72-byte save area

R1 A parameter list with the following format:

Bytes 0-3 Address of the trace record. (A standard VS1 GTF prefix followed by the trace data.)

Bytes 4-7 Address of an output buffer. (Cleared to blanks before the call.) The buffer is 80 bytes if the output is to be displayed at the terminal and 132 bytes if the output is to be printed. CSIYTXxx puts the formatted trace entry here.

Bytes 8-11 All zeros. (Not used with GCS, but put here to maintain compatibility with VS1. In VS1, this shows the address of GTF options in effect.)

Bytes 12-15 Address of the GTF Event Identifier (EID).

Bytes 16-19 Address of the trace record's data portion.

Bytes 20-23 Address of the end of the trace record's data portion + 1.

Byte 24 Flag for the following options and information:

X'40' Format the record for display on a printer.

X'80' Format the record for display on a terminal.

Byte 25 Flag byte for trace format processing.

X'01' Do not reload the formatting module for the next entry. If this bit is off then the formatting module will be reloaded on each trace entry.

Bytes 26-30 Reserved for future use.

Byte 31 A byte containing the GCS type code.

Bytes 32-63 32 byte work area for use by the called formatting routine.

When formatting the GTRACE entries, user routine CSIYTXxx should fill the output buffer at the address found in bytes 4 through 7 in the input parameter list (the address of this input parameter list is in register 1). Then it should return to the calling routine with one of the following return codes in register 15:

RC	Description
0	The user has printed the buffer and continues processing on the same GTRACE record.
4	The user has printed the buffer and is finished processing the GTRACE record.
8	Do not print the buffer and the GTRACE record is done.
Other	Print the record in hexadecimal.

Examples of Formatted External Trace Table Entries

Here are several sample supervisor event entries, as you would see them in your external trace file.

- Entry type X'02' for an IUCV external interrupt:

```
SOURCE=IUCV (type X'4000')

3D 02 useridxx   VM/GCS EXTERNAL INTERRUPT  SOURCE=IUCV
       IPPATHID, IPFLAGS1, IPTYPE = xxxx xx xx
       OLD PSW = xx x x xxxx x x xxxxxx


SOURCE=APPC/VM (type X'4000')

3D 02 useridxx   VM/GCS EXTERNAL INTERRUPT  SOURCE=IUCV
       IPPATHID, IPFLAGS1, IPTYPE = xxxx xx xx
       IPCODE, IPWHATRC, IPSENDOP = xxxx xx xx
       OLD PSW = xx x x xxxx x x xxxxxx
```

- Entry type X'03' for an I/O interrupt:

```
3D 03  useridxx   GCS I/O INTERRUPT
        CHANNEL STATUS WORD = x x xxxxxx xx xx xxxx
        OLD PSW = xx x x xxxx x x xxxxxx
```

- Entry type X'05' for an SVC interrupt:

```
3D 05 useridxx   GCS SUPERVISOR CALL INTERRUPT
        SVC CODE = xx
        TASK ID = xxxx
        FUNCTION NAME = xxxxxxxx
        OLD PSW = xx x x xxxx x x xxxxxx
```

- Entry type X'08' External Trace Table Entry by SVC GETMAIN:

```
3D 08 useridxx   VM/GCS GETMAIN VIA SVC
       TASK ID = xxxx
       KEY = xx
       SUBPOOL = xx
       STORAGE ADDRESS = xxxxxxxx
       LENGTH = xxxxxxxx
       ISSUER ADDRESS = xxxxxxxx
```

- Entry type X'09' External Trace Table Entry by SVC FREEMAIN:

```
3D 09 useridxx   VM/GCS FREEMAIN VIA SVC
       TASK ID = xxxx
       KEY = xx
       SUBPOOL = xx
       STORAGE ADDRESS = xxxxxxxx
       LENGTH = xxxxxxxx
       ISSUER ADDRESS = xxxxxxxx
```

- Entry type X'0A' External Trace Entry by Branch Entry GETMAIN:

```
3D 0A useridxx   VM/GCS GETMAIN VIA BRANCH ENTRY
       TASK ID = xxxx
       KEY = xx
       SUBPOOL = xx
       STORAGE ADDRESS = xxxxxxxx
       LENGTH = xxxxxxxx
       ISSUER ADDRESS = xxxxxxxx
```

- Entry type X'0B' External Trace Entry by Branch Entry FREEMAIN:

```
3D 0B useridxx   VM/GCS FREEMAIN VIA BRANCH ENTRY
       TASK ID = xxxx
       KEY = xx
       SUBPOOL = xx
       STORAGE ADDRESS = xxxxxxxx
       LENGTH = xxxxxxxx
       ISSUER ADDRESS = xxxxxxxx
```

- Entry type X'0C' External Trace Entry by APPC/VM Synchronous Event:

```
3D 0C useridxx   VM/GCS APPC/VM SYNCHRONOUS EVENT
       IPPATHID, IPFLAGS1, IPTYPE = xxxx xx xx
       IPCODE, IPWHATRC, IPSENDOP = xxxx xx xx
```

Here is a sample GTRACE entry as you would see it in your external trace file.

- Entry type X'0E' for a GTRACE entry:

```
3D 0E useridxx   GCS USER REQUESTED GTRACE
        TIME OF DAY CLOCK = xxxxxxxxxxxxxxxx
        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 will be dumped including the saved segments. If the dump is directed to an unauthorized user only the storage with a key of 14 and non-fetch protected storage will be dumped.

If you direct the dump to yourself or 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 non-fetch-protected storage.

You 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 will be prompted to name this common dump receiver. Choose any authorized user ID, perhaps the same user ID that you specify as your recovery machine. But make 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 (listed in "How To Initiate Dumps" on page 180) automatically will send their output to it.[9]

[9] Except for GDUMP, which optionally lets you choose another receiver.

How To Initiate Dumps

There are four different functions for requesting GCS dumps:[10]

Macros applications can issue

ABEND (with DUMP operand) This dumps the entire virtual machine as well as any saved segments (shared segments linked to your GCS system, but not within the bounds of your virtual machine). The dump automatically goes to the common dump receiver, if you have one, rather than the machine that issued ABEND.

Commands you issue

SDUMP This can dump all or part of the virtual machine. If you don't have a common dump receiver, the data goes to the machine that issued it, according to the rules of authorization in "Rules of Authorization" on page 179.

GDUMP This can dump all or part of a virtual machine's storage. You can send the dump to the common dump receiver, or the machine you issue GDUMP from, or another user ID you chose. Whatever receiver you choose, the rules of authorization in "Rules of Authorization" on page 179 apply.

SYSTEM RESTART (a #CP command) This dumps all of a virtual machine's storage, plus any saved segments, when you cannot use the GDUMP command. (Example: you have a GCS disabled loop and issue #CP SYSTEM RESTART). If you have no common dump receiver, the issuing machine gets the dump, according to the rules of authorization in "Rules of Authorization" on page 179.

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.

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.

---

[10] If you set it up beforehand, CP's VMSAVE function will also create copies of selected virtual machines either when CP itself terminates or when CP terminates them.

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.

Interactive Debugging Support

Authorized CP Commands

Authorized user IDs can have access to six CP debugging commands:

- BEGIN
- PER
- DISPLAY
- STORE
- DUMP
- VMDUMP.

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.

In addition, you should make two related CP commands, ADSTOP and TRACE, totally unavailable to both authorized and unauthorized GCS user IDs. For more information on controlling access to CP commands, see *VM/SP Planning Guide and Reference*.

Analyzing Dumps

Once storage has been dumped, it can be:

- Read in and analyzed by the receiving virtual machine under CMS with the Interactive Problem Control System (IPCS).
- Dumped to tape (using Spool-to-Tape) and sent to a Customer Engineering service team for analysis.

IPCS has some specialized routines on the IPCSDUMP and MAP commands for processing GCS dumps.[11]

To use the GCS IPCS support necessary to process a virtual machine dump the appropriate IPCS minidisks (the minidisks containing GCS and IPCS support) must be accessed before processing the dump.

A map compressing routine — compresses the GCS nucleus load map into a format IPCS can use with the IPCS MAP command. IPCS MAP will look for a nucleus load map with the default name GCSNUC MAP *. Once it is compressed, the default name is GCSIPCS MAP.

[11] For a discussion of what IPCS does and how to use it, see the *VM/SP Interactive Problem Control System Guide and Reference*.

A data extraction routine — gathers information that IPCS can use when building its problem report, like:

- Reason for the dump (GDUMP, SDUMP, SYSTEM RESTART, ABEND, program check, or "other")
- Failure location (in case of task termination or machine termination)
- Appropriate keywords (APPLADDR, CKD, COMPID, DISP, ENTRY, FAILURE, MAINTLVL, MODULE, SCPLVL, or TASKID)
- Textual data.

If this extraction routine discovers a dump in RSCSV2 format, it invokes an RSCS extraction routine to find additional problem report information.

An expanded DMMTAB communication table — includes a dump type for GCS, like the ones for CP and CMS.

A way to print formatted VTAM or VSCS control blocks — adds an option to IPCS's IPCSPRT command that lets you 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 will receive a prompt asking you if you want your dump printed using the VTAM option. If you do not pick the VTAM option you will receive a prompt asking you if you want your dump printed using the VSCS option. If you do not choose either of these options your dumps will be printed unformatted.

Subcommands for the IPCSSCAN command

For more information on IPCSSCAN subcommands see the *VM/SP Interactive Problem Control System Guide and Reference.*

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 IPCS IPCSSCAN DUMPID subcommand. This dump header will contain 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 the *VSE/VSAM Programmer's Reference.*

MM/DD/YY
Shows the date when VSAM detected the error.

HH:MM:SS
Shows 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'll find the contents of all 16 registers in the following order: registers 9-15, registers 0-8.

ABEND processing

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.

Abend completion codes give the user some idea of why the failure occurred and what area of the system may be responsible for the problem. These codes are explained in the *VM/SP System Messages and Codes.*

The abend dump contains information which enables the problem to be tracked further. Using the IPCS 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 was invoked. The ABEND work area address is located at offset X'298' in NUCON.

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:

- General purpose registers at time of failure
- PSW at time of failure
- Abend completion code and
- Reason code (if applicable).

It also contains the address of the next available trace table entry at the time the abend was invoked.

The trace table entries before this address show the events that preceded the failure.

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 (type SVC) remains on the State Block chain (see "State Block" on page 194 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'3C'	Registers at time of failure
X'40'	PSW at time of failure
X'D8'	Trace pointer at ABEND time

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.

IPCS for GCS

IPCS is a dump facility used to view dumps in VMDUMP format. The user should be familiar with IPCS and how it works before using IPCS for VM/GCS.

All dumps taken in VM/GCS will be in VMDUMP format and can be viewed using the IPCS facility. The user must have the GCS nucleus load map to use the GCS IPCS subcommands.

The IPCS component of VM/SP has some IPCSSCAN subcommands specifically oriented to display certain areas of a GCS dump.

Those IPCSSCAN subcommands are:

- 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.

Other IPCS commands may also be used with a GCS dump to aid in debugging. Any IPCS command or subcommand which is functionally categorized as "common" can be used with a GCS dump. IPCS commands such as IPCSPRT and the IPCSSCAN subcommands of CHAIN, DISPLAY, G, and LOCATE will be most helpful when debugging with IPCS.

Information Used by IPCS

General purpose control blocks are used by IPCS and can be seen in the IPCS problem report created by the IPCSDUMP command.

Those control blocks are:

- Component ID, release/service level (NUCON)
- Task ID of the currently active task (NUCON)
- Registers and PSW at the time of an abend (ABNWA) and
- Abend code (ABNWA).

For more information on Abend Work Areas see "Abend Work Area" on page 183.
Program management control blocks are displayed by IPCSSCAN subcommands.

Those fields are:

- From the Virtual Machine Load List (displayed by the VMLOADL subcommand)
  - Major NUCCBLK address
  - Module name (Major NUCCBLK)
  - Entry point address (Major NUCCBLK)
  - Module size (Major NUCCBLK)
  - Module load address (Major NUCCBLK)
  - Minor NUCCBLK address
  - Entry point name (Minor NUCCBLK) and
  - Type of Minor NUCCBLK (ALIAS or IDENTIFY).
- From the Task Load List (displayed by the TLOADL subcommand)
  - Task ID
  - Task Block address
  - Load Block address
  - Module name and
  - Load count.

Task management control blocks are displayed by the IPCSSCAN TACTIVE subcommand.

Those fields are:

- Task ID (TIDTB)
- Task Block address (TIDTB)
- Task completion code (TBK)
- State Block address (TBK and STBK)
- State Block type (STBK)
- State Block module name (STBK)
- State Block module entry point address (STBK) and
- State Block general registers (STBK).

For more information on Task Management see "Task Management" on page 193.
Storage management control blocks are displayed by the IPCSSCAN TSAB subcommand.

Those fields are:

- Chain header to the storage owned by a task (TSAB) (that is, address to the list of Minor SACBs related to the task) and
- 256 bit map of subpools owned by a task (TSAB).

In addition, the TSAB subcommand also displays for each task:

- Task ID (TBK)
- Task Block address (TBK) and
- Task Storage Anchor Block address (TBK).

For more information on Task Management see "Task Management" on page 193.

IUCV management control blocks are displayed by the IPCSSCAN IUCV subcommand.

Those fields are:

- User ID block (UIDB) address (IUCPT)
- Exit address (IUCPT)
- User word (IUCPT)
- Task Block address (IUCPT) and
- Flags of Path status (IUCPT).

For more information on IUCV see "The Path ID Table (PIDT)" on page 202.

The State of the Virtual Machine

The state of the virtual machine at the time a problem is detected can be very helpful when trying to debug the system. If the virtual machine is hung, the virtual machine state information can tell if it is running or in a wait state. If it is running, the system may be caught in a loop. If it is waiting, the VM/SP status tells for what event it is waiting. This can give a clue as to what area of the system may be causing the problem.

If the user has a CP dump, the following method may be used to obtain information about the state of the virtual machine:

- Using IPCS, issue the IPCSDUMP subcommand VMBLOK with the user ID.
- This will display the VMBLOK contents for the user, registers, interrupts pending, and last command issued. The VM/SP status is also given.

If the user is debugging interactively, the following method may be used to obtain information about the state of the virtual machine:

- Issue the CP command LOCATE with the user ID to get the address of the VMBLOK for the user.
- Using the address of the VMBLOK, issue the CP command DCP for address VMBLOK + X'58' for 4 bytes.
- The VM/SP running status is given in the first byte.

Byte	Field description
X'80'	waiting console function
X'40'	waiting paging operations
X'20'	waiting scheduled IOBLOK start
X'10'	virtual PSW wait
X'08'	waiting instruction simulation
X'04'	user not logged on
X'02'	user logging off
X'01'	idle wait state

Figure 12. VM/SP Running Status

- If the waiting instruction simulation bit is on, issue the CP command DCP for address VMBLOK + X'98' to see what instruction is being simulated.
- If the instruction operation code is X'B2F0', it is an IUCV instruction.

For more information on VMBLOK see *VM/SP CP Data Areas and Control Blocks*, (LY24-5220).

NUCON and SIE

The GCS NUCON and SIE are control blocks located in the first virtual page of GCS. Each GCS virtual machine, when logged on, has its own NUCON and SIE.

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 (PSW)
- Channel Status Word (CSW) (X'40' in the NUCON)
- Channel Address Word (CAW) (X'48' in the NUCON)
- Virtual machine's user ID (X'204' in the NUCON)
- Task ID of the currently active task (X'212' in the NUCON).

In addition, other important GCS control blocks are pointed to by NUCON fields.

Examples of those control blocks are:

- Task Block of the currently active task (pointed to from X'214' in the NUCON)
- Common Trace Block (pointed to from X'21C' in the NUCON)
- SIE (pointed to from X'5C4' in the NUCON) and
- Various work areas (as, 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:

- Address of Task ID Table (X'010' in SIE)
- Address of the Asynchronous Exit Queue (X'018' in SIE) and
- Address of the Virtual Machine Control Block (VMCB) (X'02C' in SIE).

Virtual Machine Control Block

When a virtual machine IPLs GCS, a Virtual Machine Control Block (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:

- Virtual machine user ID (first 8 bytes of the VMCB) and
- Machine ID (2 bytes at displacement X'0A' of the VMCB).

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, as 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. Display the storage contents at X'5C4' by issuing

   ```
   CP D 5C4.4
   ```

2. Add X'28' to the hexadecimal value that is displayed.

The result is the storage location of the beginning of the list of VMCBs.

The same procedure can be used in IPCS with the IPCSDUMP DISPLAY subcommand.

An alternate method to get the address of the list of VMCBs is to find the address of the module CSIOME. You can find this address in the GCSNUC MAP, or by issuing the IPCSSCAN MAPN subcommand (MAPN CSIOME). That address points to the beginning of the list of VMCBs.

How to Locate the GCS Common Lock

The Common lock is pointed to by the SIELKCOM field in the SIE (at displacement X'20'). The common lock is just a word in length, located in common storage, and contain the machine ID (2 bytes) and task ID (2 bytes) that is 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, you can do so with the CP PER command. You can issue a PER on store into the common lock word, and when PER 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 IPCSSCAN 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 PER 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. NUCON starts at location X'000'. The following are the mapping of the important fields in the NUCON and SIE.

NUCON

Displacement	Field description
X'000'	New PSW
X'008'	Old PSW
X'010'	CVT Address
X'014'	BGCOM Address
X'018'	External Old PSW
X'020'	SVC Old PSW
X'028'	Program Check Old PSW
X'030'	Machine Check Old PSW
X'038'	I/O Old PSW
X'040'	Channel Status Word
X'048'	Channel Address Word
X'054'	Address of Table Trace Header
X'058'	External New PSW
X'060'	SVC New PSW
X'068'	Program Check New PSW
X'070'	Machine Check New PSW
X'078'	I/O New PSW
X'080'	SYSCOM Address
X'204'	Virtual Machine User ID
X'20C'	Release/Service Level
X'212'	Task ID of Active Task
X'214'	Address of Active Task
X'21C'	Common Trace Block Pointer
X'298'	Abend Workarea Address
X'2E8'	Command Input Line
X'388'	Tokenized Plist
X'588'	Extended Plist
X'5C4'	SIE Address
X'5DC'	DEB Entry Chain Address
X'5E4'	NUCCBLK Chain Address
X'650'	FCB Chain Address
X'654'	Number of FCBs in Chain
X'674'	Address of DEVTAB
X'678'	Address of ADTSECT
X'67C'	Address of DIODA
X'680'	Address of AFTSTART
X'688'	Total Virtual Machine Time (DIAGNOSE code X'70')
X'690'	Time of day when dispatched (DIAGNOSE code X'70')
X'6A0'	Global loadlibs name list address
X'6A4'	Global loadlibs directory list address
X'6A8'	Size of Global name list and directory list
X'6AC'	Number of Global loadlibs

GCS Console Constants

Displacement	Field description
X'6DC'	Attention Interrupt ECB
X'6E0'	I/O complete ECB
X'6E4'	Output pending ECB
X'6E8'	Command task ECB
X'6EC'	Console task flag

Displacement	Field description
X'80'	Read I/O in process
X'40'	Write I/O in process
X'20'	Attn pending
X'10'	Output pending

X'6F0'	Pointer to first command input buffer
X'6F4'	Pointer to last command input buffer
X'6F8'	Pointer to first WQE buffer on queue
X'6FC'	Pointer to last WQU buffer on queue
X'700'	Pointer to first ORE buffer on queue
X'704'	Pointer to last ORE buffer on queue

SI Extension

Displacement	Field description
X'10'	Address of Task ID Table
X'14'	Address of First Task Block in Dispatch Queue
X'18'	Address of Asynchronous Exit Queue
X'20'	Address of Common Storage Lock
X'24'	Task Id waiting for Lock
X'27'	Program Management Flags
	Byte **Field description**
	X'80' Global Loadlib Issued
	X'40' OSRUN is Active
X'28'	Address of VMCB Array
X'2C'	Address of This Machine's VMCB
X'30'	Address of VSAM Sysnames Table
X'40'	Address of SMAB
X'54'	Attn Interrupt ECB
X'58'	I/O Complete ECB
X'5C'	Console Output Pending ECB
X'60'	Command Task ECB
X'64'	Console Task Flags
X'68'	Address of First Command Input Buffer
X'6C'	Address of Last Command Input Buffer
X'70'	Address of First WQE Buffer on Queue
X'74'	Address of Last WQE Buffer on Queue
X'78'	Address of First ORE Buffer on Queue
X'7C'	Address of Last ORE Buffer on Queue
X'80'	Console CCW (Read/Write)
X'88'	Second CCW (No-op)
X'90'	Bit String of ORE Ids
X'9D'	Last Id used for Assigning OREs
X'A0'	Address of Trace Anchor Block (TAB)
X'A4'	Address of Nucleus Extension Control Block Chain
X'B8'	Address of IUCV Anchor Block
X'BC'	Signal System Services Path ID
X'C0'	Address of Start of Available Common Free Storage
X'C4'	Address of Start of Available Private Free Storage
X'C8'	Size of this Virtual Machine
X'CC'	Address of TQE Pool
X'D4'	Flags
	Byte **Field description**
	X'02' Machine is Authorized
	X'01' Machine has 4K pages
X'D5'	System Save Time
X'DD'	System Save Date
X'E8'	Highest Priority Task that is Ready for scheduling
X'EC'	Time Slice for Tasks (in Microseconds)

For more information on NUCON and SIE see Appendix D, "GCS Control Blocks" on page 257.

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).
- Value of the registers and PSW when the task is dispatched, if the task is dispatchable.
- The address of the task block of the mother task.
- The task ID and task priority.

The Task Block contains the following information:

Displacement	Field description
X'00'	Task on dispatch queue of higher priority
X'04'	Task on dispatch queue of lower priority
X'08'	Next task on dispatch queue of same priority
X'0C'	Previous task on dispatch queue of same priority
X'10'	Active State Block address
X'14'	Address of Load List
X'18'	PSW
X'28' to **X'64'**	Registers
X'88'	Mother Task Address
X'90'	Subtask Address
X'A4'	Machine Id
X'A6'	Task Id
X'A8'	Address of Task Storage Anchor Block (TSAB)
X'AC'	Address of IUCV EIB chain
X'C4'	Task completion code (ABEND)
X'C8'	Abend reason code
X'CA'	Task Storage Key
X'CB'	Dispatching Priority
X'CD'	Flags
	Byte / **Field description**
	X'80' Task in problem state
	X'40' Independent application
	X'20' Task has terminated
	X'02' Dump requested by abnormal termination
X'D8'	Time Task was dispatched

State Block

State Blocks (STBLK) are used by GCS 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 to 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 completes execution, 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 program running under the previous (execution-wise) State Block. The PSW for a running program is in the Task Block.

The State Block contains the following important fields:

Displacement	Field description	
X'00'	Program name	
X'08'	PSW of the caller	
X'10'	Address of the next state block on chain	
X'14'	Address of the previous state block on chain	
X'18'	Address of the task block for this state block chain	
X'1C'	Address of the NUCCBLK for this program	
X'20'	Entry point of program or SVC	
X'24'	FLAGS	
	Byte	**Field description**
	X'80'	Link Block
	X'40'	SVC Block
	X'20'	Asynchronous Exit Block (AEB)
X'25'	FLAGS	
	Byte	**Field description**
	X'20'	AEB for Scheduled Exit
	X'10'	AEB for General I/O
	X'08'	AEB for Timer
X'26'	Wait Count (1 byte)	
X'30' to X'6C'	Callers register save area (all registers)	

WAIT COUNT Field in a State Block

An important field in a State Block is called WAIT COUNT. This field (STBWAIT at displacement X'26' in the State Block) allows you to determine if a task is waiting.

If the field contains:

- 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 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 (or ECBLIST) associated with the wait. This allows you 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. This address allows you 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. This address allows you to determine the module that has issued the SVC instruction.

AEB Block

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 block, indicate if 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, entry point of the exit, and then 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 and 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," the task "A" issues the GCS SCHEDEX macro, specifying the task ID of task "B" and the exit 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, and before dispatching any tasks, it checks if there are any AEBs queued in the SIEAEQ.

If so, it will then take the AEB off the SIEAEQ and queue it at the beginning of task "B" State block chain.

4. When task "B" eventually gets dispatched, the exit will be run as the currently active State Block.

VMCSCHDX Is a field in the Virtual Machine Control Block (VMCB) and 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 Appendix D, "GCS Control Blocks" on page 257. An example how this queue is used is:

1. When a task "A" in the virtual machine "A" wants to schedule an exit to run in a task "B" in the virtual machine "B," the task "A" issues the GCS SCHEDEX macro, specifying the Machine ID of virtual machine "B," the task ID of task "B," and the exit address.
2. GCS SCHEDEX processing, running for the "SCHEDEXing" 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 about the exit to be scheduled.
4. The virtual machine "B" is interrupted by the IUCV message (External interrupt).
5. The IUCV interrupt handler in GCS calls the GCS scheduling routines CSISDT and CSISDX.

   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," and before dispatching any tasks, it checks if there are any AEBs queued in the SIEAEQ.

   If so, it will then take the AEB off the SIEAEQ and queue it at the beginning of task "B" State block chain.
7. When task "B" eventually gets dispatched, the exit will be run 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 (SVC's), 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 will be given control to run first.

To keep track of tasks and their priorities, a dispatch queue is set up which chains the tasks (via 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 will switch to another task of equal priority that is waiting in the dispatch queue. This will only happen 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 will be dispatched in a round robin method.

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 IPCS and Task management control blocks see "IPCS for GCS" on page 184.

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.

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 table entries start at the TIDTB address + X'08'.
- 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.
- Each TIDTB has 255 entries.

Each TIDTB entry describes a task:

- Each entry is 8 bytes long.
- The first halfword (first 2 bytes) in an entry is the task ID.
- The following halfword (second 2 bytes) is unused.
- The next fullword (last 4 bytes) is the address of the task block for that task.

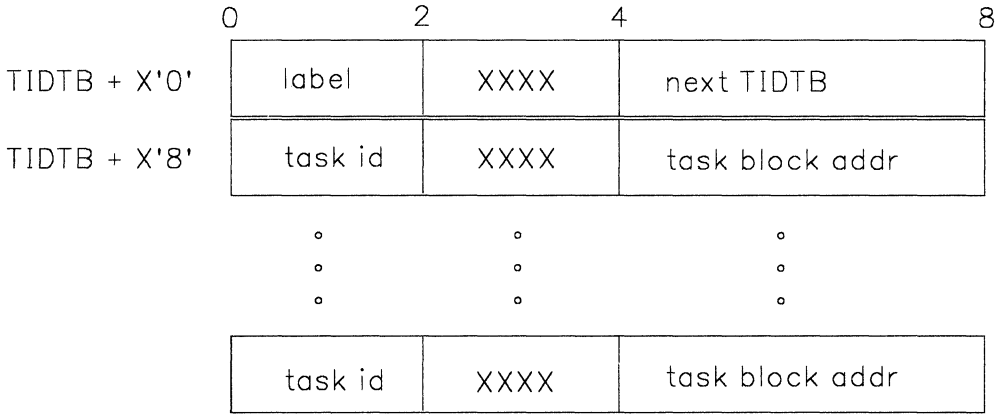


Figure 13. Task Id Table (TIDTB)

How to find which task is running

In NUCON there is a field which contains the task ID of the task which is currently running. Use this task ID and find its entry in the task ID table. In NUCON there also is a field which 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'.

Refer to the "State Block" on page 194 and "Task Block" on page 193 for important fields.

If using IPCS, the following procedure using IPCSSCAN subcommands will yield similar information in formatted form:

- Issue DISPLAY X'212' to get the current task ID.
- Issue TACTIVE using the task ID just found.
- The display which results includes the completion code, program name, and register contents associated with the state block.

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 re-dispatched, 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 which issued the GTRACE.

IUCV

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:

- Other routines in same machine (same task or different task),
- Routines in other virtual machines, or
- 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 invokes the IUCV CONNECT function via the IUCVCOM macro, and the target communicator invokes the IUCV ACCEPT function, again via 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, via the PRIV=YES parameter, if running in supervisor state. This allows the task to communicate on a path using IUCV directly, rather than through the GCS IUCV support.

For more information on IUCV see the *VM System Facilities for Programming*. GCS IUCV support is further discussed in the *VM/SP Group Control System Command and Macro Reference*.

Note: In the IUCV section, when the word user appears, it refers to any supervisor or problem program.

Applications Debugging

When IUCV problems are first suspected, the debugger should ensure that the application or program running is using IUCV correctly and the parameter lists are set up correctly. PER 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, the failure 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.

The IUCV Anchor Block (IUCAB)

The IUCV anchor block (IUCAB) 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 information:

Displacement	Field description
X'00'	EIB address
X'04'	UIDB address
X'08'	IUCV parameter list address
X'0C'	PIDT address
X'10'	Max number of paths

The first address in the IUCAB points to the external interrupt buffer (EIB) which contains information about the last IUCV external interrupt which was processed by GCS.

Pointers to the User ID Block (UIDB) chain and the Path ID Table (PIDT) are also found in the IUCAB. These control blocks are explained further in the their own sections.

The IUCV parameter list address points to the internal parameter list used by GCS IUCV support. The internal parameter list is used to hold 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 must be initiated by the GCS IUCV support. For example, to SEVER an incoming path to a user that has not issued an IUCVINI SET function.

The User Id Blocks (UIDB)

User ID blocks contain information about active users in the IUCV environment. There is a UIDB for each user, containing the user name, user word, and associated task block address. The UIDBs are chained together with the most recently added user at the beginning of the chain. The first UIDB is pointed to from the IUCV anchor block (IUCAB) at IUCAB + X'04'.

The user ID block is built when the user is admitted into the IUCV environment via 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 terminates 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'	Next user ID block address
X'04'	General Exit address
X'08'	User Name
X'10'	User word
X'14'	Task block address

X'18'	Flags	
	Byte	**Field description**
	1xxx	Problem state indicator
	x1xx	Privilege state indicator

The Path ID Table (PIDT)

The path ID table contains an entry for every possible IUCV path based on the maximum number of paths allowed for this virtual machine. A path entry is filled in when the path is established via a 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 IPCS and IUCV management control blocks see "IPCS for GCS" on page 184.

Each path ID table entry is 20 (X'14') bytes long.

The Path ID Table contains the following information:

Displacement	**Field description**	
X'00'	Address of User Id Block	
X'04'	Exit Address	
X'08'	User word	
X'0C'	Task block address	
X'10'	Flags	
	Byte	**Field description**
	X'80'	path active
	X'40'	connect issued
	X'20'	connect pending
	X'10'	path is quiesced
	X'08'	path is severed

The task block address represents the task which 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

Information about a path, such as who owns it and its present status, is found in a path ID table entry for the path. The path ID is used to index into the path table to get to the entry that describes the particular path.

- If the debugger has a VMDUMP formatted dump, the IPCS facility can be used.
  - Issue the IPCS IPCSSCAN IUCV subcommand.
  - The resulting display will show the important information found in each of the path entries in the path ID table.
- If the debugger is manually displaying addresses and following chains, the following procedure will yield 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 (IUCAB) address at SIE + X'B8'.
- Locate the path ID table (PIDT) address at IUCAB + X'0C'.
- The specified path ID is in hexadecimal.
- Calculate the offset.

```
OFFSET =  path ID  x  X'14'.
    (Each path table entry is X'14' or 20 bytes long)
    EXAMPLE:  path ID = X'B'
    path entry is at displacement X'B' x X'14' = X'DC'
    into the table.
```

- Path entry = PIDT + offset.
- See path ID table entry map for 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)
- Storage protection (Storage key and fetch or store protection bits)
- Storage ownership (persistent or task related storage).

Storage Anchor Blocks

To find the Storage Anchor Blocks, locate the SI extension (SIE) address in NUCON currently at X'5C4'. At X'40' in the SI extension is a pointer to the Storage Management Anchor Block (SMAB). In the SMAB there are pointers to private and common Storage Anchor Blocks. The Private Storage Anchor Block (PSAB) is pointed to by the SMAPSAB pointer at SMAB + X'00'. The Common Storage Anchor Block (CSAB) is pointed to by the SMACSAB pointer at SMAB + X'04'. The fields in both Storage Anchor Blocks are identical. Obtain the appropriate pointer for the Storage Anchor Block requested.

The are three types of Storage Anchor Blocks:

- Private Storage Anchor Blocks (PSAB)
- Common Storage Anchor Blocks (CSAB)
- Task Storage Anchor Blocks (TSAB).

The PSAB and CSAB contain pointers to the start of arrays of Major and Minor SACBs (Storage Anchor Control Blocks) that describe the free storage pages.

The TSAB contains a pointer to the start of an array of Minor SACBs that describe the storage belonging to a task. The TSAB is pointed to by the field TBKSTOR (at displacement X'A8') in the Task Block. To find the PSAB and CSAB:

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 PSAB is pointed to by the SMAPSAB field (at SMAB + X'00').
4. The CSAB is pointed to by the SMACSAB field (at SMAB + X'04').

The Storage Anchor Blocks contain the following fields:

Displacement	Field description
X'04'	ANCHKEYP (in PSAB or CSAB) starts an array of 32 pointers that are the anchors for chains of Major SACBs chained by key.
X'84'	ANCHPGMN (in PSAB or CSAB) points to the first page of Minor SACBs. Each page of Minor SACBs include forward and backward pointers to other pages of Minor SACBs.
X'88'	ANCHPGL (in PSAB or CSAB) points to the Major SACB that describes the lowest completely free page of storage.
X'8C'	ANCHPGH (in PSAB or CSAB) points to the Major SACB that describes the highest completely free page of storage.
X'90'	ANCHMAJL (in PSAB or CSAB) points to the Major SACB that describes the lowest free page of storage.
X'94'	ANCHMAJH (in PSAB or CSAB) points to the Major SACB that describes the highest free page of storage.

Description of the SACBs

There are two types of Storage Anchor Control Blocks (SACBs):

- Major SACBs and
- Minor SACBs.

Both are 16 bytes long. The combinations of the Major and Minor SACBs describe a page of free storage. Each Major SACB is followed in contiguous storage by a Minor SACB.

Major SACBs are control blocks used to describe a page of free storage. There is one Major SACB per page of free storage. They are found in contiguous storage, are built at initialization time, and are permanent.

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 a length field of 2 bytes that gives the largest free area on the page, that does not begin on a page boundary.
X'0A'	MAJLNCON a length field of 2 bytes that gives the length of the free area at top of the page.
X'0C'	MAJKEY an 8-bit field that contains the key and fetch bit for the page.
X'0D'	Flags

Byte	Field description
X'10'	MAJBLK is a flag in the Major SACB (fifth to the low-order bit in the byte X'0D' into the Major SACB) that is used to mark a page of a large block of storage.

X'08'	MAJBKFST is a flag in the Major SACB (fourth to the low-order bit in the byte X'0D' into the Major SACB) that is used to mark the first page for a large block of storage.
X'04'	MAJBKLST is a flag in the Major SACB (third to the low-order bit in the byte X'0D' into the Major SACB) that is used to mark the last page for a large block of storage.
X'02'	MAJENDL is a flag in the Major SACB (second to the low-order bit in the byte X'0D' into the Major SACB) that is used to mark the Major SACB for the lowest page of storage.
X'01'	MAJENDH is a flag in the Major SACB (low-order bit in the byte X'0D' into the Major SACB) that is used to mark the Major SACB for the highest page of storage.

Important fields in Minor SACBs

Minor SACBs are control blocks used for two purposes:

- 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.

The Minor SACBs contain the following fields:

Displacement	Field description
X'00'	MNORNXT points to the next minor SACB used to describe the next non-contiguous 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 four bytes.

- Used to 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 non-contiguous 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 four bytes

- Used to describe storage allocated by a task. These minor SACBs are found in a chain pointed to by the TSAB which is pointed to by the task block.

Displacement	Field description
X'00'	MNORNXT points to the next Minor SACB used to describe storage allocated by the task.
X'04'	MNORPTRF points to the allocated storage.
X'08'	MNORLN is the length of the allocated storage, this field has a length of 4 bytes.

For more information on IPCS and Storage management control blocks see "IPCS for GCS" on page 184.

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 then a page, scan the Major SACB between these major SACBs to see if there is sufficient storage.

Scanning the Major/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 address of page represented (MINPTRF at X'04').
   c. Match page represented with address of storage in question.
      1) These minor SACBs are contiguous with Major SACBs they describe.
      2) Scroll until corresponding page is found.

Checking free storage on any given page

1. Find the appropriate anchor block for 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/Minor SACBs."
3. The Minor SACB is the header for a chain of Minor SACB that describe all free storage for the page. This first Minor SACB describes the free storage on the lower page boundary. IF MNORLN is 4K then the page is fully free and is available for use in any key.
4. If MNORLN is not 4K then look at MAJMAXLN. This field will tell you the largest free piece of storage available on the page not on a page boundary.

   Note: Since 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.

The Minor SACBs use the following fields:

Displacement	Field description
X'00'	MNORNXT is the pointer to the next Minor SACB
X'04'	MNORPTRF is the pointer to the start of the free area
X'08'	MNORLN is the length of the free area

Finding the key for a given page

- To find the actual key for a given page of storage use the CP command DISPLAY K.
- How to check 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/Minor SACBs" on page 206.
  2. Find the key and fetch bit in MAJKEY.
     a. 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 non-fetch protected storage.

MAJKEY | 000kkkkF | CP KEY | KKKKFXXX |

- Checking 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 PSAB or CSAB) is the start of an array of 32 pointers that are the anchors for chains of Major SACBs for each key and protection status.
  3. 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 non-fetch, the second for key zero fetch protected etc.
     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.

How to find the storage belonging to a given task

1. Find the task block.
2. Find TBKSTOR (X'A8' into the task block) which points to the Task Storage Anchor Block (TSAB).
3. TSAOWNED (X'24' into the TSAB) points to a chain of Minor SACBs belonging to the task.
4. Fields in the Minor SACB and their meaning:

Displacement	Field description
X'00'	MNORNXT is the pointer to the next Minor SACB
X'04'	MNORPTRF is the pointer to the allocated storage
X'08'	MNORLN is the length of the allocated storage that describes additional allocated storage owned by the task

How to check what subpools belong to a given task

1. Find the task block.
2. Find TBKSTOR (X'A8' into the task block) it points to the TSAB.
3. TSASPOOL (X'04' into the TSAB) is a 256 bit map of all possible subpool values. Each subpool number that is owned by the task will have the appropriate bit on. If the bit is off then there is some mother task with the corresponding bit on. You can chase up the task chain to find the owner of any given subpool by looking for the appropriate bit to be on for some mother task. At least one task will have the bit on. The commands task has all 256 bits on.

Common Storage Management Problems

Freemain or Getmain go 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 only 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 "mother" task that owns the subpool.).
   c. Find TBKSTOR (X'A8' into the task block). It points to the Task Storage Anchor Block (TSAB).
   d. TSASPOOL (X'04' into the TSAB) is a 256-bit map of all the subpools owned by this task. Either the active task or one of the mother 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.
2. The chain of Major SACBs describing the Free Storage have been broken or the bit marking the highest or lowest major SACB has been turned off causing a search off the end of the Major SACB. (See MAJENDL and MAJENDH.) Scan the Major SACB and check to see if they have been overlaid. (Follow instructions for "Scanning the Major/Minor SACB.")

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.).
   a. If the length is valid then check for fragmentation (See "Checking for Storage Fragmentation" on page 206.)
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 like 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 off 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 the allocated storage and free storage can be found on pages of minor SACBs pointed to by ANCHPGMN found in the anchor blocks. Thus the minor SACBs can also be easily scanned.

f. Check the trace table for the last Getmains. See if Freemains are done for that storage.

Abend 778:

1. Invalid mode byte in SVC parameter list
2. Program returning storage in wrong key
   a. Returning someone else's storage
   b. Privileged program could have changed the key
3. Storage management ran out of storage for internal control blocks.

Things to check:

1. Check the parameter list set up by the macro.
2. Check to see if 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 207.
3. Check for fragmentation.

Tracing Storage Management

Supervisor tracing using ITRACE and ETRACE include the tracing of GETMAINs and FREEMAINs (invoked by SVC or by request via branch entry) as they occur in GCS. GETMAIN trace entries (X'08' type for SVC and X'0A' for request via branch entry) and FREEMAIN trace entries (X'09' type for SVC and X'0B' for request via branch entry) contain much of the same information:

- The task ID
- Storage address obtained or released
- Length of the storage
- Storage subpool and
- 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 for DASD and console is performed using the GCS GENIO macro. However, as GCS does not provide any device specific code, the use of the GENIO macro requires that the application that requests the I/O has to perform all the related I/O control tasks, including error recovery.

The GCS GENIO macro is used 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 getmains 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 Is used to close 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 212 for a discussion of the GIOTB table).

The exit routine specified in the GENIO OPEN macro will no longer be scheduled if I/O interrupts are received from the device.

MODIFY Is used to modify a CCW of an active I/O program. DIAGNOSE code X'28' is issued to CP to effect the CCW modification.

CHAR Is used to request the characteristics (such as, device class, type, model and so on) of a device.

GCS gets this information by using DIAGNOSE code X'24'.

The CHAR function does not require the device to be open to obtain the requested information.

START Is used to initiate 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 executed 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 a previously initiated operation.

GCS issues a virtual SIOF instruction to the device.

If the virtual channel or control unit is busy, the I/O operation is queued on a virtual channel queue. When a "channel end" or a "control unit end" interrupt is received, the START function is tried again.

Another START function to the device is not accepted until the current operation completes. The end of the operation is identified by device end interrupt.

STARTR Is used to allow an authorized program to 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 being the fact that GCS uses DIAGNOSE code X'98' instead of a SIOF 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.

HALT Is used to force GCS to issue a HALT DEVICE instruction to the specified virtual device address.

General I/O in GCS allows a program to drive any I/O device that is defined on the virtual machine except DASD. Using the GENIO macro a user can obtain, use, and release any I/O device. For further information on the GENIO Macro, refer to *VM/SP Group Control System Command and Macro Reference*.

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 the 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 is used as a save area when I/O interrupts are being handled.

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 and the CSW are saved in this control block.
  - The PSW contains the address of the interrupting device in the PSW interrupt code (the second half word of the PSW).
  - The CSW contains the condition code, unit status, channel status, and channel command word address.
- Contains a 16-by-16 array of address words that allow you to find the GIOTB entry associated with a specific open device.
- Contains a pointer to the Page Fix Table (PFXTB) that allows you to find the pages that have been locked in real storage.
- Contains the anchor blocks for the Virtual Channel Queue (VCHQ) that allow you to find the I/Os outstanding.
- Contains the control unit index and channel index that allow you to determine the address of the device which last had GENIO processing done (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 CSIIOSAV in the GCS nucleus map.

The IOSAVE contains the following information:

Displacement	Field description
X'090'	I/O old PSW
X'098'	CSW from I/O causing interrupt
X'0A0'	General I/O table addresses
X'4A0'	Address of Page Fix Table
X'4A8'	Channel index into General I/O table
X'4AA'	Control unit index into General I/O table
X'4AC'	Addresses for start of virtual channel queues
X'52C'	Real I/O authorization flag

The control unit index and channel index stored in the IOSAVE gives the address of the device which last had GENIO processing done, either from an interrupt or issuing a GENIO macro. The saved PSW and CSW are stored in the IOSAVE from the last I/O interrupt. Refer to the *System 370 Reference Summary* or *System 370 Principle of Operations* for the mapping of the PSW and CSW.

The General I/O Table (GIOTB)

The General I/O table (GIOTB) is found at IOSAVE + X'A0'. It is a 16-by-16 array of addresses that are indexed by channel and control unit. A device address contains the channel and control unit addresses. The device address is two bytes long. Half of a byte is called a nibble. The first nibble of the device address is not used. The second nibble is the channel address. The third nibble is the control unit address. The fourth nibble is the device address for that channel and control unit.

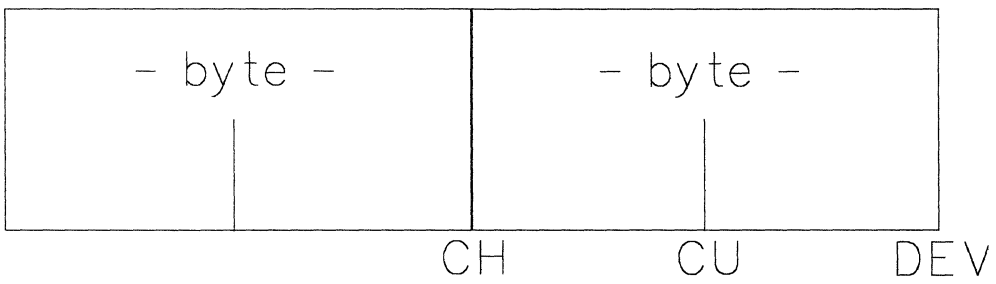


Figure 14. Device Address

The General I/O table (GIOTB) contains a GIOTB entry for each open device.

A GIOTB entry provides information about the device, such as:

- Device Address.
- 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 terminates.

- 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 at GIOTB + X'34', contains information related to the exit and will be enqueued on the AEB queue pointed to by the SIE at SIE + X'18'.

- Characteristics of the device (virtual and real).

The General I/O table (IOSGIOTB) in the IOSAVE control block contains 256 indexes to point to a GIOTB entry for every Channel/Control Unit combination. The field IOSGIOTB is found at IOSAVE + X'A0'. To find the GIOTB entry for a specific device:

- Convert channel (CH) and control unit (CU) addresses to decimal.
- Calculate the offset into the table. Offset = (CH x 64) + (CU x 4).
- Convert the offset into hexadecimal.
- Add the hexadecimal value and the address of the beginning of the GIOTB in the IOSAVE to obtain the address for the first GIOTB entry for the devices for that channel and control unit.
- The GIOTB entries for the devices on the channel and control unit are chained together. The address for the next GIOTB entry in the chain is found at GIOTB + X'00'.

The General I/O table contains the following information:

Displacement	Field description
X'00'	Address of next entry in table
X'04'	Device address
X'08'	Address of task requesting open
X'0C'	Task ID of task requesting open
X'0F'	Flags

Byte	Field description
X'80'	I/O active
X'40'	I/O queued
X'20'	Synchronous interrupt queued
X'10'	Exit scheduled for synchronous interrupt
X'08'	Asynchronous interrupt queued
X'04'	Asynchronous interrupt pending
X'02'	Wait

X'14'	Address of exit when I/O has been completed (GIOEXIT)
X'18'	Characteristics of virtual device
X'1C'	Characteristics of real device
X'24'	Address of CCW to be started
X'30'	Virtual channel queue element
X'34'	Address of Asynchronous event block (AEB)
X'3C'	Synchronous interrupt control block (ICB)
X'64'	Asynchronous interrupt control block (ICB)

I/O Interrupt Handling

When an I/O interrupt indicates "channel end" or "control unit end," the appropriate virtual channel queue is checked.

If there is an I/O request queued, that I/O is restarted. If the I/O is successfully restarted, the I/O request is dequeued from the virtual channel queue.

If "channel busy" or "control unit busy" is received again, the I/O request remains on the virtual channel queue.

If there is an error on the restart, the request is dequeued and the channel queue is searched to see if there is another I/O operation which can be restarted.

Table 4 indicates the actions that occur depending on the type of interrupt received for a GENIO device.

Table 4. I/O Interrupt Action Table

Type of Interrupt	GCS Action
Terminating error conditions (unless channel end given without DE)	Schedule Exit Search Virtual Channel Queue for I/O to be Restarted.
Channel and Device End	Schedule Exit Search Virtual Channel Queue for I/O to be Restarted
Device End	Schedule Exit Search Virtual Channel Queue for I/O to be Restarted
Channel End	Search Virtual Channel Queue
Control Unit End	Search Virtual Channel Queue
Asynchronous Interrupt	Schedule exit

The exit specified in the GENIO OPEN macro is provided with the CSW from the interrupt, and sense bytes if a unit check occurred. If a CSW is sent prior to the "device end" CSW (for example "channel end" is received before "device end"), the first CSW is saved. When subsequent CSWs are received, the status bytes from the CSW are ORed with the CSW already stored in the interrupt control block.

The exit receives control in the key and state of the task that opened the device.

- If the task is an authorized program, the exit is entered with interrupts disabled.
- If the task is not an authorized program the exit is entered with interrupts enabled.

Interrupt Control Blocks

Within each GIOTB entry are two Interrupt Control Blocks (ICB) which 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 in the first byte.

The ICBs contain the device address, and CSW.

The Interrupt Control Blocks contain the following information:

Displacement	Field description
X'04'	Device address
X'08'	CSW from interrupt
X'10'	Sense bytes (synchronous only) (24 sense bytes)

Virtual Channel Queue

One of General I/Os responsibilities is to queue I/O requests when a channel or control unit is busy. Queuing is done on a channel level. Since there are 16 channels for I/O, there are 16 Virtual Channel Queues for requests for I/O.

The field IOSVCHQ in the IOSAVE control block contains 16 x 2 pointers to the 16 Virtual Channel queues. (each pair points to the beginning and to the end of each Virtual Channel queue). The field IOSVCHQ is found at IOSAVE + X'4AC'.

Each Virtual Channel queue is a FIFO queue:

- When an I/O operation for a device is queued, the GIOTB entry associated with that device is added to the end of the queue.
- When a channel is available for I/O, GCS selects the first GIOTB entry in the queue, and issues the I/O operation to the corresponding device.

  If the I/O is successful, that GIOTB entry is dequeued from the beginning of the queue. Otherwise, the next GIOTB entry is selected and the process is repeated.

How to Find the I/O Queued for a Channel

One of General I/O's responsibilities is to queue up I/O requests when an I/O channel or control unit is busy. Queuing is done on a channel level. Since there are 16 channels for I/O, there are 16 queues for requests for I/O. Pointers for these queues are found in IOSAVE starting at IOSAVE + X'4AC'. Each entry in IOSAVE for a channel contains a pointer to the beginning of the queue and the end of the queue. When I/O is queued, a GIOTB entry is added to the end of the queue. When the channel is available for I/O, the next I/O request begins processing and the GIOTB entry is dequeued from the beginning of the queue.

To find the I/O requests queued for a specific channel (CH):

1. Find the address of the IOSVCHQ field of IOSAVE.

   ```
   IOSVCHQ = IOSAVE + X'4AC'
   ```

2. Find the pointers to the first and last GIOTB entry associated with the channel (CH).

   Each entry in the IOSVCHQ field in IOSAVE is 2 words long:

   - 1st word = address of first GIOTB entry in the queue
   - 2nd word = address of last GIOTB entry in the queue.

   ```
   FIRSTENTRY = IOSVCHQ + ( CH x 8)
   LASTENTRY = FIRSTENTRY + 4
   ```

For example:

- IOSAVE+X'4AC' = first GIOTB entry queued for CH0
- IOSAVE+X'4B0' = last GIOTB entry queued for CH0
- IOSAVE+X'524' = first GIOTB entry queued for CHF
- IOSAVE+X'528' = last GIOTB entry queued for CHF.

3. The GIOTB entries for the other devices with I/O requests on that channel are chained from this GIOTB entry by the GIOVCHQE field in each GIOTB entry (at displacement X'30').

How to find what pages are locked by PGLOCK

The Page Fix Table (PFT) is used to keep 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 the IOSAVE (IOSAVE + X'4A0').

The IOSAVE contains the following information:

Displacement	Field description
X'00'	Address of next PFT entry
X'04'	Virtual address of page
X'08'	Real address of page
X'0C'	Task ID that locked page

How to find the characteristics of a device

The GENIO macro with option CHAR will give information about a specific device. The data returned will contain both real and virtual characteristics. The device does not have to be open for the user to issue 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 will be zero.

I/O Debugging

I/O problems can occur in one of 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 SNA LU terminal or printers hanging, a VTAM link will not initialize, or a questionable status is returned from I/O.

When the user suspects an I/O problem, the first thing that should be done is to keep track of error messages and keep the console log, especially for VTAM. I/O problems generally require recreating the problem using traces. Traces can be set 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.

1. The following sets up traces for CP, GCS, VSCS, and VTAM:
   a. CPTRAP ALLOWID VTAM (where VTAM is the *vmid*)
   b. CPTRAP START TO *userid*

c. CPTRAP 3D (collect virtual machine group trace data)
d. VSCS TRACEON (EXT (start VSCS external trace)
e. ETRACE GTRACE SIO I/O {GROUP}
f. VTAM F TRACE, ID=*luname*, TYPE=BUF or TYPE=I/O (start VTAM trace).

2. Recreate the problem.
3. To turn off the traces:
   a. CPTRAP STOP (stop CP trace)
   b. VTAM F NOTRACE, ID=*luname*, TYPE=BUF or TYPE=I/O (stop VTAM trace)
   c. ETRACE END (stop GCS trace)
   d. VSCS TRACEOFF (stop VSCS trace).

Trace Table Entries

After tracing has been completed, the trace events for all of the areas which were traced are found in the GCS internal trace table, unless wrap has occurred. If GCS is using an external trace, the trace entries will be in the CPTRAP spool file. VTAM and VSCS entries in the trace table will be entered as GTRACE entries.

GCS tracings of SIO and I/O contain helpful information for debugging.

- The SIO trace entry includes the CAW which will contain the channel program address. This address should be checked to ensure that a valid channel program is being invoked.
- The I/O trace entry includes the CSW and I/O old PSW at the time of the I/O interrupt. Channel and program status should be checked at the time of the interrupt.

The CP TRACE for SIO and I/O will trace virtual and real I/O passed to CP for processing. The CP trace entries will reflect the information given to CP for the I/O operations.

- The TRACE SIO command traces TIO, CLRIO, HIO, HDV, and TCH instructions for all virtual devices. The trace entry resulting from this command includes such information as the condition code and CSW.
- The TRACE I/O command traces virtual machine I/O interrupts. It does not trace channel-to-channel I/O operations. With the CSW trace option also specified, the resulting trace entry will include the contents of the virtual and real CSW at the I/O interrupt.

For more information on debugging VTAM see the *VTAM Diagnosis Guide*, (SC23-0116).

Recreating the Problem

When unexpected results occur on terminals or other SNA devices, the problem should be recreated with VTAM and VSCS traces on. This will help to isolate the component at fault. Most hung LU conditions will not be GCS problems. They will probably be 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.

The following gives checkpoints during problem recreation where helpful information about I/O can be obtained and how to set the checkpoints:

1. When Real I/O is being issued, a PER stop should be set to see the state of the system, what was sent, and the channel program used.
   - Issue CP TRACE I/O command to trace real I/O in CP.
   - Set a PER I DATA 83. (83 is the operation code for the DIAGNOSE instruction.)
   - When the PER stop is encountered, the fullword operation code is displayed. If the fourth byte is a X'98', it is Real I/O. Otherwise, issue BEGIN to continue.
   - If the DIAGNOSE is for Real I/O, then Real I/O has just been done.
   - Display the PSW to check the interrupt code and condition code.
   - The CP TRACE facility will display the CSW used in the real I/O just processed.
   - Display the CCW at the address given in the CSW. It contains the data address used in the Real I/O.
   - Display the CAW at X'48'. It contains the channel program address.
   - This information will help to determine what the GCS I/O request was.
2. If the I/O requested is a READ, the read buffer address should be noted before the I/O is issued. After the I/O is done, display the read buffer to see what CP has passed back to GCS as a result of the read.
3. When I/O is being tracked for a VTAM application, the user should look at the parameter list which is being passed to GCS in the GENIO macro.
   - Set a PER stop at the beginning of the GCS GENIO module (CSIGIM). This address is found in the Load Map for GCS.
   - When VTAM issues the GENIO macro for I/O processing, the PER 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 via commands. The GCS supervisor and the applications can communicate with the operator via 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 via the CMDSI macro. The CDMSI macro allows the user to issue GCS, CP, or LOADCMD defined commands from within a program running in GCS. For more information on the CMDSI macro refer to the *VM/SP Group Control System Command and Macro Reference.*

LOADCMD Command

The LOADCMD command is included in the command support. LOADCMD allows a user to define their own command name 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, refer to the *VM/SP Group Control System Command and Macro Reference.*

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, then 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 which 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.

The NUCON contains the following command areas:

Displacement	Field description
X'2E8'	Command input line
X'388'	Tokenized parameter list
X'3B8'	Address of command token
X'3BC'	Address of beginning of argument string
X'3C0'	Address of end of argument string
X'5C4'	Address of SIE

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 will be included in the extended parameter list.

SIE Information

The SIE also contains a commands and console area. This commands and console area contains such information as ECBs, CCWs, and pointers to the queues for the commands, the messages, and the replies which have not yet been processed.

The SIE contains the following command and console areas:

Displacement	Field description
X'54'	Attention Interrupt ECB
X'58'	I/O Complete ECB
X'5C'	Output pending ECB
X'60'	Command ECB
X'64'	FLAGS
	Byte **Field description**
	X'80' Read I/O in progress
	X'40' Write I/O in progress
	X'20' Attention pending
	X'10' Output pending
X'68'	Address of first CMDBUF on queue
X'6C'	Address of last CMDBUF on queue
X'70'	Address of first WQE on queue
X'74'	Address of last WQE on queue
X'78'	Address of first ORE on queue
X'7C'	Address of last ORE on queue
X'80'	Read/Write CCW
X'88'	No-Op CCW
X'90'	ORE ID bits
X'9D'	Last assigned ORE ID

Each ECB in the SIE is one word or 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.

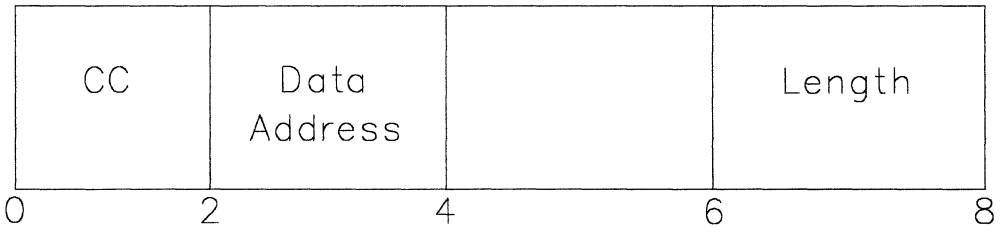
Three queues are maintained by the communications task, they are:

- CMDBUF
- Write Queue Elements (WQE) and
- Operator Reply Elements (ORE).

Each of these queues are pointed to from within the SIE and contain elements which 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 contain two CCWs. The first CCW is used for READ/WRITE, the second CCW is a no-op. The CCW contains a command code (CC), data address, and 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 CCW is mapped as follows:



CC = X '0A' ⇨ READ
CC = X '09' ⇨ WRITE

Figure 15. CCW mapping

The ORE ID bits in the SIE are used to keep track of which reply numbers are outstanding (00-99). 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 issued and will not be 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 will contain the same information as is 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'	next CMDBUF on queue
X'04'	length of command data
X'08'	command input data
X'8C'	address of command token
X'90'	address of start of argument string
X'94'	address of end of argument string
X'B0'	tokenized parameter list

WQE and ORE

The WQE queue consists of messages to the VM/SP 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'	Address of next WQE on chain
X'06'	Length of message text
X'08'	Message text

An ORE contains the following information:

Displacement	Field description
X'00'	Address of next ORE on chain
X'04'	Reply ID
X'08'	Address of task block which issued message
X'0C'	Length of message issued text
X'10'	Message issued text
X'8C'	Key of issuer
X'8D'	Length of reply
X'90'	Address of reply buffer
X'94'	Address of reply ECB

A user can see if a message has not been processed yet 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 issue the QUERY REPLY command which will return all messages which 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 product.

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 utilize a DOS segment.
- GCS includes basic support for VTAM.

- SET SYSNAME command can only be used before the VSAM environment is initialized in GCS.
- GCS associates open ACBs with the task which performed the open. When a task terminates, 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 will concentrate on those areas in VSAM support which are unique to GCS or have been changed from CMS. The debugger should have some knowledge of how VSAM works in CMS and GCS and the differences. More information on GCS support of VSAM is found in the *VM/SP Group Control System Command and Macro Reference*. General information on VSE/VSAM support within VM/SP is found in the *VM/SP Application Development Guide for CMS*, *VM/SP CMS Data Areas and Control Blocks*, and *VM/SP CMS Diagnosis Reference*.

NUCON Changes

The GCS NUCON differs from the CMS NUCON in regard to VSAM support. The following gives a summary of the changes in the NUCON for GCS support of VSAM and other information which is still found in the NUCON.

- The Communications Vector Table (CVT) address is still located at X'10' in the NUCON. No changes have been made in the CVT.
- 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 been changed for GCS:

Displacement	Field description
X'20'	Address of VSAM anchor block - 1
X'3B'	Dump option flag which is always set
X'8C'	Flag for GETVIS area initialized

- The System Communications Region (SYSCOM) address which is located at X'4E4' in the CMS NUCON is located at X'80' in the GCS NUCON. No changes have been made to the SYSCOM.

VAD Information

The VTAM/VSAM Data Block (VAD) is a new data area added for GCS support of VSAM. This data block resides in the first 64K 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 the execution 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 workareas chain, open ACBs list, and DOSCB chain.

The VAD contains the following information:

Displacement	Field description
X'04'	Address of 1st VSAM workarea
X'08'	Address of start of VSAM segment
X'10'	Address of start of BAM segment
X'18'	Address of 1st DOSCB
X'1C'	Addresses of VTAM routines
X'28'	Address of VSE transient area
X'30'	Address of VSE lock table
X'34'	Address of simulated VSE TCB
X'38'	Address of VSE ppsave area
X'3C'	Address of VSE LTA save area
X'40'	Number of DOSCBs in effect
X'88'	Address of list of open ACBs
X'8C'	Length of open ACBs list
X'90'	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 invoked prior to giving control to GCS/VSAM.

VSAM Anchor Block

In GCS, the anchor block contains only the addresses of the VSAM AMCB table and OAL table. 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 Workareas

A VTAM/VSAM workarea (VIPWORK) is established for each GCS task running VTAM/VSAM. The workareas are chained together with the newest task VIPWORK added to the beginning of the chain. VIPWORKs are removed from the chain when the task terminates.

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 located at VIPWORK + X'50'.

The VIPWORK contains the following information:

Displacement	Field description	
X'50'	Address of next VIPWORK	
X'54'	Address of previous VIPWORK	
X'58'	Address of Temporary OPEN/CLOSE ACB list	
X'5C'	Size of Temporary OPEN/CLOSE ACB list	
X'5E'	Task Id	
X'7E'	Flags	
	Byte	**Field description**
	X'80'	PSW condition code = 0
	X'40'	PSW condition code = 1
	X'20'	PSW condition code = 2
X'80'	Savearea for Callers Registers	
X'BC'	VIP Entry Caller Return Address	
X'F0'	DOS Return Code to User	

Helpful Hints for VSAM debugging

The following are GCS commands and macros which can be used to give 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 via the SET SYSNAME command.
- DLBL - without any operands specified will display the current file definitions as were defined by the DLBL command.
- SHOWCB - macro which will return the fields of a specified control block within VSAM.
- TESTCB - macro which will test the values in the fields of a specified control block within VSAM.
- IDUMP - VSAM IDUMP macro is supported by GCS. GCS will convert the request to an SDUMP macro for processing.

Chapter 7. 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 229
- Using dumps, described in "Using TSAF Dumps to Diagnose Problems" on page 229
- Using system trace data, described in "Using System Trace Data to Diagnose Problems" on page 232.

In addition, "Interactive Service Queries" on page 234 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 the 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 must do the following steps:

1. Collect information about the error.
   - Save the console sheet 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 via the CP VMDUMP command (described in the *VM/SP CP General User Command Reference*). CP sends the dump to TSAF's virtual reader.
   - Save any CPTRAP file 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 failure 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, TSAF then issues a CP SYSTEM RESET command. If the CONCEAL option is on, as recommended, CP automatically IPLs CMS. Otherwise, you, the operator, must re-IPL CMS. Similarly, if TSAF is not invoked from the PROFILE EXEC, you must restart the TSAF virtual machine.

VM/SP System 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 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 *VM/SP CP General User Command 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 16 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 16. Sample TSAF Console Log

Using TSAF Dumps to Diagnose Problems

You can use IPCS 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.

The steps involved in using dumps to diagnose problems are:

1. Create a TSAF IPCS map, if it does not already exist.
2. Create the TSAF dump.
3. Process the TSAF dump.
4. Diagnose the TSAF dump.
5. Print the TSAF dump.

Creating the TSAF IPCS Map

Note: You only need to do this step when a new CMS or TSAF nucleus is built.

When a new CMS or TSAF nucleus is built, enter the following IPCS command to compress the TSAF load map for IPCS:

```
map tsaf
```

The default name for the map source file is TSAF MAP; and the default name for the input CMS nucleus load map is CMSNUC MAP. The default name for the compressed map file is TSAFIPCS MAP, which you create using MAP TSAF.

Note: If you do not have the compressed map file, IPCS facilities, which allow for diagnosis with dumps, are greatly reduced. For instance, without the map you could not invoke the formatted display subcommand (FDISPLAY) and you would not receive any TSAF control block information.

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 the TSAF virtual machine cannot create the dump, you can use the VMDUMP command. The VMDUMP command dumps virtual storage that VM/SP creates for the virtual machine user; in this case, for TSAF. The dump goes to the virtual machine specified by the SYSDUMP parameter on the SYSOPR macro in the DMKSYS ASSEMBLE file, if you enter the following CP command:

```
vmdump 0-end system format tsaf
```

Do not use the reserved names of ATSCAB1 or ATSCAB2 for the dump ID of VMDUMP. The *VM/SP CP General User Command 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 IPCS command:

```
ipcsdump
```

The default map file is TSAFIPCS MAP.

When you issue IPCSDUMP, it invokes a TSAF routine to extract information from the dump and transmit it to IPCS for inclusion in the problem report and/or symptom summary. IPCSDUMP creates the following:

- Problem report
- Symptom summary
- Disk resident dump to which IPCS appends the map information.

See the *VM/SP Interactive Problem Control System Guide and Reference* for more information about the IPCSDUMP command.

Diagnosing a TSAF Dump

The IPCSDUMP command generates a symptom record, which is based on problem report information. 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:
  - Error code
  - ID of the failing component
  - ID of the failing module
  - Registers and PSW contents.

You can also use the IPCS IPCSSCAN command to examine the dump interactively. The IPCSSCAN command is described in the *VM/SP Interactive Problem Control System Guide and Reference*. The following sections introduce those subcommands specifically for TSAF (FDISPLAY and TRACE).

Note: TRACE can also be used for CP dumps.

Displaying the TSAF Dump Information

The FDISPLAY subcommand of the IPCSSCAN 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 BSC or LINKCTL CTCA).

See the *VM/SP Interactive Problem Control System Guide and Reference* for a complete listing of the FDISPLAY parameters and for some example outputs of the FDISPLAY subcommand.

Formatting and Displaying Trace Records

TSAF maintains an internal trace table within the TSAF virtual machine. You can use the TRACE subcommand of IPCSSCAN 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. See the *VM/SP Interactive Problem Control System Guide and Reference* for examples of using the TRACE subcommand and the sample outputs.

You can also scroll through the formatted or hexadecimal output with either of the following IPCSSCAN subcommands:

- TRACE SCROLL or SCROLLU
- SCROLL or SCROLLU.

See the *VM/SP Interactive Problem Control System Guide and Reference* for more information about the IPCSSCAN TRACE and SCROLL subcommands.

Printing a TSAF Dump

The IPCSPRT command prints the dump and symptom record that IPCSDUMP processed. The output you get consists of the following:

- Symptom record
- Dump in hexadecimal (no special formatting)
- Appended load maps
- Contents of the registers and the PSW.

See the *VM/SP Interactive Problem Control System Guide and Reference* for more information on the IPCS IPCSPRT command.

Using System Trace Data to Diagnose Problems

TSAF maintains an internal trace table within the TSAF virtual machine. You can use the IPCSSCAN TRACE subcommand to display the internal trace table entries. TSAF also writes trace entries to the system CPTRAP file. You can then use IPCSSCAN to view TSAF entries.

Setting External Tracing

The TSAF SET ETRACE command lets you enable or disable external tracing for the TSAF virtual machine. If you want to collect TSAF trace records, issue the following from the TSAF virtual machine before CPTRAP is started:

```
set etrace on
```

When you set external tracing on, certain internal TSAF trace records are written externally to a CPTRAP spool file. A complete description of the SET ETRACE command is in the *VM/SP Connectivity Planning, Administration, and Operation.*

Using CPTRAP to Trap Trace Table Entries

The CPTRAP command collects TSAF information in a reader file. This information helps with problem determination.

Note: CPTRAP is a privileged CP command. In most installations, the TSAF virtual machine is not given the necessary privilege class to be able to issue the CPTRAP command. For this reason, the command must be issued by some other virtual machine that has the authority to do so.

The following commands activate CPTRAP for TSAF records only:

`cptrap id` *tsafl* `type gt allowid` *userid* `3e`

userid is the TSAF virtual machine user ID. The 3E operand selects 3E entries that the TSAF virtual machine produces to be spooled by the CPTRAP facility.

`cptrap enable id` *tsafl*

Enter:

`cptrap close`

to end CPTRAP processing. When you issue this command, the CPTRAP SPOOL file goes to your reader.

For more specific information about the CPTRAP command, see the *VM/SP CP System Command Reference* and "Debugging with the CPTRAP Facility" on page 95.

Viewing CPTRAP data with IPCS

Refer to the *VM/SP Interactive Problem Control System Guide and Reference* for specific details on how to view CPTRAP data.

Trace Table Entry Format for TSAF

The trace table entries vary in length and follow the format described below. The length fields are one-byte long and may be any number from 0 to 255. The length and data fields are optional data fields.

A trace table entry looks like the following:

Table 5. TSAF Trace Table Entry

length(1)	data(1)	. . .	length(n)	data(n)	TRAILER RECORD

The trailer record format looks like the following:

Table 6. TSAF Trace Table Trailer Record				
Clock (STCK format)	Characters 4 through 6 of module name	Trace ID code	Data area length	'E00E'x

The lengths associated with each field 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 LINK displays information about the links that TSAF currently has.
- 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.

See the *VM/SP Connectivity Planning, Administration, and Operation* for more specific information about the TSAF QUERY command.

Chapter 8. Debugging AVS

Effective problem diagnosis for APPC/VM VTAM Support (AVS) is a multi-step process consisting of:

- Dump analysis
- System trace data analysis
- Use of the AVS QUERY command
- 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 the system programmer or other person specifically responsible for diagnosing system problems.

Using AVS Dumps to Diagnose Problems

The Interactive Problem Control System (IPCS) analyzes dumps and tracks problems in VM/SP. You can use IPCS to collect and diagnose problem data for the AVS virtual machine. Because AVS runs in a GCS group, GCS and AVS subcommands for the IPCSSCAN command can be issued when a dump is processed.

The steps used to diagnose problems using dumps are:

1. Obtain a GCS IPCS map, if one doesn't already exist
2. Obtain the AVS dump
3. Process the AVS dump
4. Use IPCSSCAN to diagnose the AVS dump.

Obtaining the GCS IPCS 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, issue the following to compress the GCS load map for IPCS:

```
map gcs
```

Refer to the *VM/SP Interactive Problem Control System Guide and Reference* for more information on how to compress the GCS load map.

If you do not have the GCS load map, the GCS subcommands for the IPCSSCAN command will be affected. The AVS subcommands for the IPCSSCAN command will be unaffected.

Creating an AVS Dump

When a problem occurs due to an abend, or when an abnormal condition is detected, AVS produces a dump.

There are two types of dump produced by AVS:

- A dump is produced when AVS abends.
- A problem dump is produced when the system detects an error but does not cause AVS to abend.

The problem dump takes a snapshot of the system at the time that the problem occurred; this is a first-time data capture on the failure. An informational message appearing at the operator console corresponds to a message number generated with the problem dump report.

The frequency of problem dumps is determined by the operator, with a default amount of 20. The MAXPROBD field in the AVSTUN ASSEMBLE file is used to optionally specify the number of dumps taken from the point at which AVS is started.

Users wishing to create a dump for the AVS machine should enter:

```
gdump 0-end format avs dss
```

The GDUMP command dumps virtual storage that VM/SP creates for the virtual machine; in this case, for AVS. The *VM/SP Group Control System Command and Macro Reference* contains more information about the GDUMP command.

Processing an AVS Dump

To load any AVS virtual machine dump directly onto a disk, issue the following:

```
ipcsdump
```

The default map file is GCSIPCS MAP.

When you issue IPCSDUMP, it invokes an AVS routine to extract information from the dump and transmit it to IPCS for inclusion in the problem report and/or symptom summary. IPCSDUMP creates the following:

- Problem report
- Symptom summary
- Disk resident dump to which IPCS appends the map information.

Refer to the *VM/SP Interactive Problem Control System Guide and Reference* for more information about the IPCSDUMP command.

Diagnosing an AVS Dump

The IPCSDUMP command generates the symptom record, which is based on problem report information. 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:
  - Error code
  - ID of the failing component
  - ID of the failing module
  - Registers and PSW contents.

You can also use the IPCSSCAN command to examine the dump interactively. The IPCSSCAN command is described in the *VM/SP Interactive Problem Control System Guide and Reference*. The following sections introduce those subcommands specifically for AVS (GDISPLAY and TRACE).

Displaying the AVS Dump Information

The GDISPLAY subcommand of the IPCSSCAN command displays data control blocks and addresses important to the AVS virtual machine. You can get information about the following by invoking different GDISPLAY parameters.

- Scheduling global block (SGB)
- Global control block (GCB)
- Conversation block (CVB)
- Remote LU block (RLU)
- Gateway block (GWB)
- Subtask control block (SCB)
- Module addresses (MAPN)
- Module names (MAPA)
- Gateway parameters (GWBPTRS)

Refer to the *VM/SP Interactive Problem Control System Guide and Reference* for more information on the GDISPLAY subcommand.

Formatting and Displaying Trace Records

AVS maintains an internal trace table within the AVS virtual machine. The TRACE subcommand of IPCSSCAN formats and displays trace records from the AVS internal trace table. Using the HEX or FORMAT parameters, you can display the trace table entries in hexadecimal or formatted display. See the *VM/SP Interactive Problem Control System Guide and Reference* for examples using the TRACE subcommands with sample outputs.

You can also scroll through the formatted or hexadecimal output with either of the following IPCSSCAN subcommands:

- TRACE SCROLL or SCROLLU
- SCROLL or SCROLLU

See the *VM/SP Interactive Problem Control System Guide and Reference* for more information about the IPCSSCAN TRACE and SCROLL subcommands.

Using System Trace Data to Diagnose Problems

AVS maintains an internal trace table within the AVS virtual machine. The IPCSSCAN TRACE subcommand displays internal trace table entries. AVS also writes trace entries to the system CPTRAP file. You can then use IPCSSCAN to view the CPTRAP file.

Setting Internal Tracing

To trace AVS events internally or externally, you must turn on internal tracing. To turn on internal tracing, use the AGW SET ITRACE ON command. Selectivity on internal tracing may be done on a gateway basis. Internal tracing information is written to an internal wrap table in the AVS virtual machine. If you want to collect AVS trace records internally, issue the following from the AVS virtual machine:

```
agw set itrace on
```

A complete description of the SET ITRACE command is in *VM/SP Connectivity Planning, Administration, and Operation.*

Setting External Tracing

The AGW SET ETRACE command allows you to 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. If you want to collect AVS trace records, issue the following from the AVS virtual machine after CPTRAP is started:

```
etrace gtrace
agw set etrace on
```

When you have internal and external tracing set on, AVS trace records are written externally to a CPTRAP spool file. A complete description of the AVS SET ETRACE command is in *VM/SP Connectivity Planning, Administration, and Operation.*

Using CPTRAP to Trap Trace Table Entries

The CPTRAP command collects AVS information in a virtual reader file. This collected information assists in problem determination.

Note: Because the AVS virtual machine is not set up to diagnose problems, only one authorized user (class C user ID) at a time may issue the CPTRAP command.

The following commands activate CPTRAP for AVS records:

`cptrap id` *trapid* `type gt allowid` *userid* `3f`
(*trapid* is the name of this trap; *userid* is the AVS virtual machine user ID.)

`cptrap enable id` *trapid*

This activates CPTRAP. Enter:

```
cptrap stop
```

to end CPTRAP processing. When you issue this command, the CPTRAP SPOOL file goes to your reader.

For more specific information about the class C CPTRAP command, see the *VM/SP CP System Command Reference* and "Debugging with the CPTRAP Facility" on page 95.

Getting Information about CPTRAP with QUERY

The QUERY CPTRAP command obtains the following information about CPTRAP. For example,

- QUERY CPTRAP ALL displays the status of the traps.
- QUERY CPT TT ALL returns a table that shows the current selectivity (on, off, or extra) for each type of CPTRAP record recorded in the trace table and CPTRAP file.
- QUERY CPTRAP TYPE GT displays the status of a GT-type trap ID.

Entering the following sequence of commands (requesting trace information for VMBLOK FF3AE8):

```
cptrap id ex1 type tt intable alloff
cptrap id ex1 intable 5 on infile 5 on
cptrap id ex1 intable c on infile c vmblok ff38ae8
cptrap id ex1 intable 6 off infile 6 off
query cptrap type tt 05 06 0c
```

results in the following:

```
ID = EX1      DISABLED
TYPE=TTABLE   SET=NULL

INTABLE 05:ON
INFILE 05:ON
INTABLE 06:OFF
INTABLE 06:OFF
INTABLE 0C:ON
INTABLE 0C:VMBLOK FF3AE8
```

Note: **INFILE** indicates the selected CPTRAP file; **INTABLE** indicates the selected trace tables.

Entering CPTRAP STOP ends CPTRAP processing. For more information about the class C QUERY command, see the *VM/SP CP System Command Reference.*

Viewing CPTRAP Data with IPCSSCAN

The IPCSSCAN command is used to review the external trace entries written to CPTRAP. After issuing IPCSSCAN to get into the CPTRAP file, select the AVS records you wish to view by entering:

```
select 3d code 0e
```

This will select all AVS external trace records from the CPTRAP file. The entries can then be formatted using the trace and scroll subcommands.

Note: Any other products running under GCS and writing to the CPTRAP file will also have their trace records selected for formatting. For more information about the IPCSSCAN command, refer to the *VM/SP Interactive Problem Control System Guide and Reference.*

Trace Table Entry Format for AVS

AVS trace table entries vary in length and follow the format described below. The length fields are one byte long and may be any number from 0 to 236. An AVS TRACE entry cannot exceed 255 bytes. The length and data fields are optional.

Table 7. Trace Table Entry

length(1)	data(1)	...	length(n)	data(n)	TRAILER RECORD

The trailer record format is described here:

Table 8. Trace Table Trailer Record				
Clock (STCK format)	Characters 4 through 6 of module name	Trace ID code	Data area length	'E00E'x

The lengths associated with each field 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.

Interactive Service Queries

The AVS query command provides information about the operating AVS virtual machine.

- AGW QUERY GATEWAY displays the gateway names that are currently in the collection.
- 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 ITRACE displays the current setting of the internal tracing.
- AGW QUERY ALL displays all of the above information.

Refer to *VM/SP Connectivity Planning, Administration, and Operation* for more information about this command.

Summary of Steps to Follow When an AVS Abend Occurs

When an AVS abend occurs, the following actions are required:

- 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.
  - Issue the MAP command to convert the GCS load map to a format that allows IPCSDUMP to append the GCS load map to the dump.
  - Issue the IPCSDUMP command to process the dump in question. Study the problem report produced by IPCSDUMP processing.

  - Issue the IPCSSCAN command with the necessary subcommands to look at the contents of the dump.
  - Save any CPTRAP file that contains AVS data (described in "Using System Trace Data to Diagnose Problems" on page 238).

- 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 via the CP VMDUMP command (described in the *VM/SP CP General User Command Reference*).

VM/SP System Messages and Codes lists the various AVS abend codes and their causes.

Appendixes

- Appendix A: Problem-Specific Checklists
- Appendix B: Control Registers
- Appendix C: Stand-Alone Dump Formats
- Appendix D: GCS Control Blocks

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 dump.
5. A CP nucleus loadmap.
6. 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).
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. Any output generated that demonstrates the problem.
3. The contents of the PSW. (Take particular note of PSW bits 40 through 63. A CP wait state code might be stored there.)
4. The contents of the general registers.
5. A CP restart dump.
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).
11. If available, the wait state code.

Checklist for Incorrect or Unexpected Output from an Application Program

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. The contents of the PSW.
2. The contents of the general and control registers.
3. The contents of storage locations from hexadecimal address 00 to 100.
4. If possible, the instructions (and their addresses) that are involved in the loop.
5. Any console or printed output that demonstrate the problem.
6. A CP restart dump.
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).
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. Control Registers

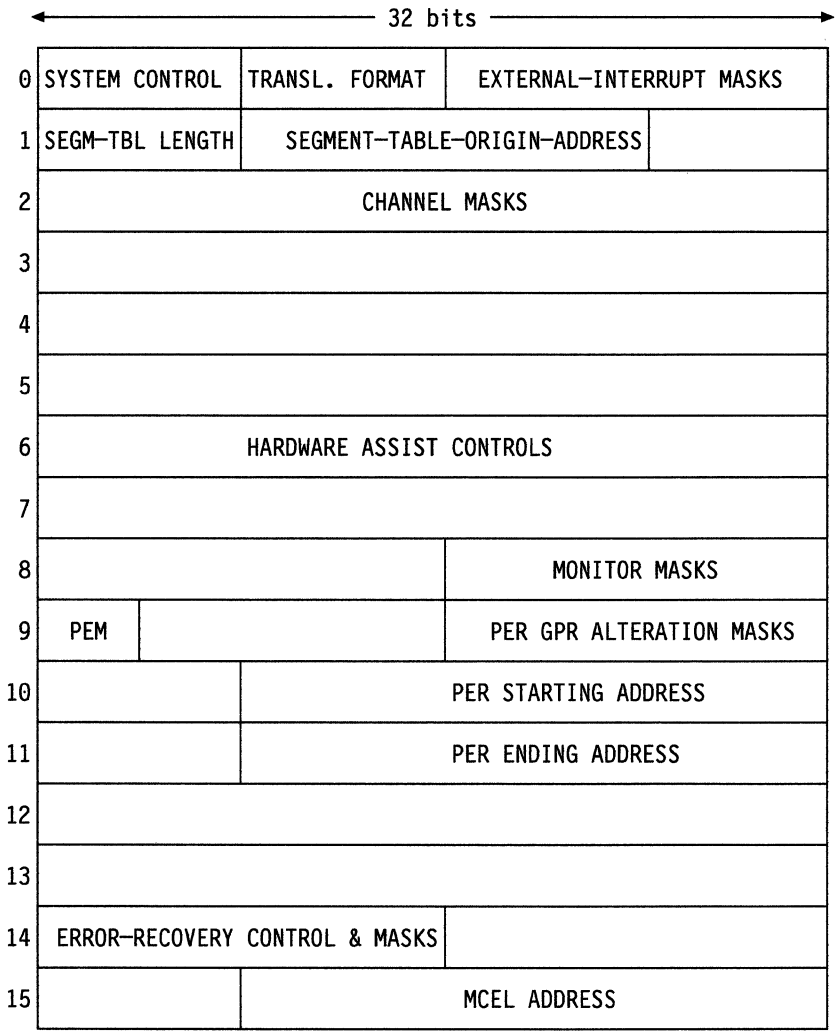
The control registers are used to maintain and manipulate control information that resides outside the Program Status Word (PSW). There are sixteen 32-bit registers for control purposes. The control registers are not part of addressable storage.

At the time the registers are loaded, the information is not checked for exceptions, such as invalid segment-size or page-size code or an address designating an unavailable or a protected location. The validity of the information is checked and the errors, if any, indicated at the time the information is used.

Figure 17 on page 250 is a summary of the control register allocation.

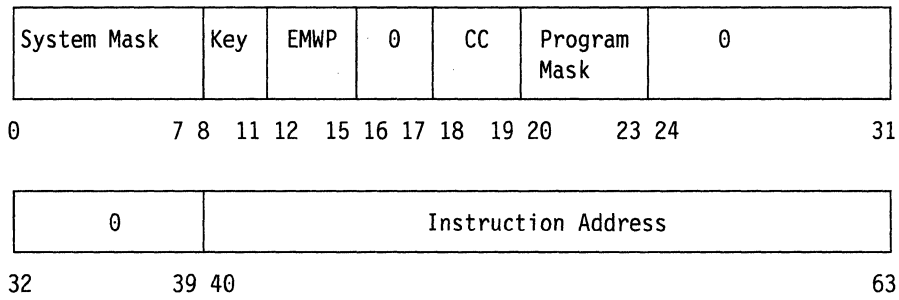
Figure 18 on page 251 is a description of the Extended Control (EC) PSW.

For more information, refer to the *IBM System/370 Principles of Operation*, GA22-7000.



PEM = PER EVENT MASKS

Figure 17. Control Register Allocation



The fields of the EC MODE PSW are:

Bits	Contents
0	Must be zero.
1	PER (Program Event Recording) enabled.
2-4	Must be zero.
5	Address translation.
6	Summary I/O mask.
7	Summary extension.
8-11	The protection key determines if information can be stored or fetched from a particular location.
12	Extended control mode.
13	The machine check flag is set to 1 if machine check interruptions are enabled.
14	The wait state flag is set to 1 when the CPU is in the wait state.
15	The problem state flag is set to 1 when the CPU is operating in the problem rather than the supervisor state.
16-17	Must be zero.
18-19	The condition code reflects the result of a previous arithmetic, logical, or I/O operation.
20-23	The program mask indicates whether or not various program exceptions are allowed to cause program interrupts.
24-39	Must be zero.
40-63	The instruction address gives the location of the next instruction to be executed for program interrupts or of the instruction last executed for external interrupts.

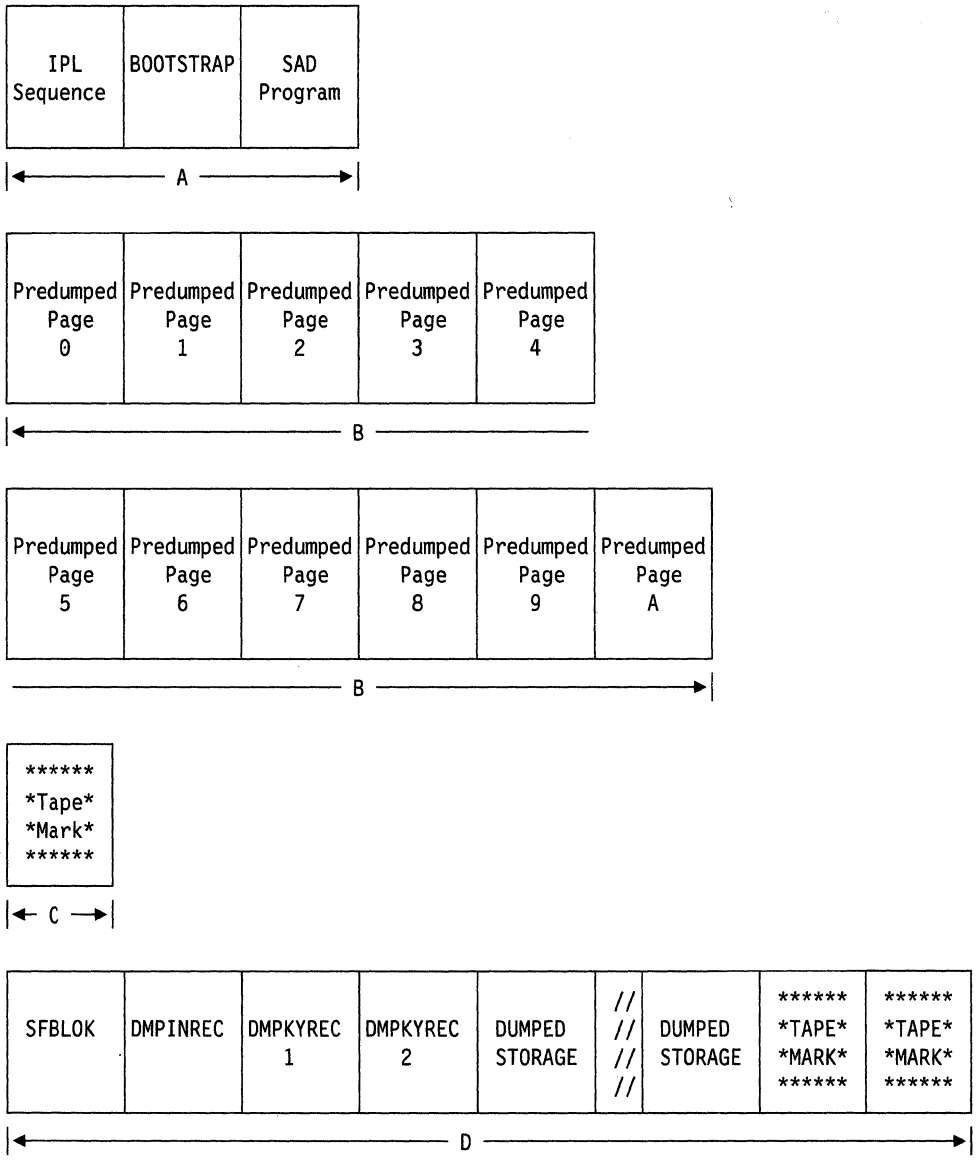
Figure 18. The Extended Control PSW (Program Status Word)

Appendix C. Stand-Alone Dump Formats

Tape Format

A tape used with the stand-alone dump facility has the format shown in Figure 19 on page 254.

- If the IPL tape and the dump device are not the same, the IPL tape includes sections A, B, and C.
- If the dump device is a tape, but not the same tape as the IPL tape, the dump output tape includes sections C and D.
- If the IPL device and the dump device are the same, the tape includes sections A, B, C, and D.



A = = > Written by the Stand-Alone Dump Utility on the IPL tape at generation time.

B = = > Written by BOOTSTRAP on the IPL tape.

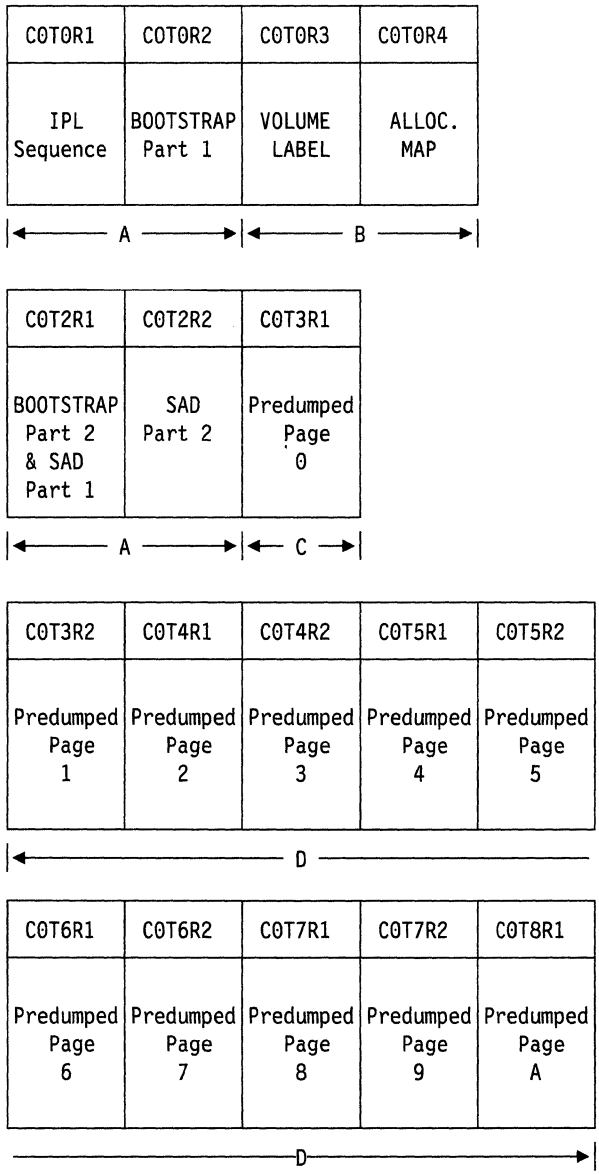
C = = > Written by BOOTSTRAP if the IPL tape is the same as the dump tape. Written by stand-alone dump program if the IPL tape is not the same as the dump tape.

D = = > Written by the stand-alone dump program on the dump tape.

Figure 19. Stand-Alone Dump Facility Tape Format

DASD Format

When you use a DASD device to IPL the stand-alone dump program, the system uses cylinder 0 to hold the program. Cylinder 0 must be CP formatted and allocated as permanent space. The stand-alone dump facility has the format shown in Figure 20.



A = = > Written by the Stand-Alone Dump Utility on the IPL DASD at generation time.

B = = > Written by FORMAT/ALLOCATE program.

C = = > Written by BOOTSTRAP Part 1 on the IPL DASD.

D = = > Written by BOOTSTRAP Part 2 on the IPL DASD.

CnTnRn identifies cylinder, track, and record numbers.

Figure 20. Stand-Alone Dump Facility DASD Format

Printer Format

Dumps to printer devices are printed as follows:

- CP formats the following data fields for each processor, beginning with the processor where the stand-alone dump program was IPLed:
  - CPU address (only if in AP or MP mode)
  - General purpose registers
  - Control registers
  - Floating point registers
  - Clock comparator
  - CPU timer values
  - Stored-status PSW
  - Prefix value (only if in AP or MP mode)
  - External interrupt old/new PSWs
  - SVC old/new PSWs
  - Program check old/new PSWs
  - Machine check old/new PSWs
  - I/O interrupt old/new PSWs.
- The following fields are printed for the processor where the stand-alone dump program was IPLed:
  - TOD clock.
- Lines of duplicate data will have a suppression message after the first line of the data is printed.
- A half page (2048 bytes) of all zeros has one line of zeros printed with the key, followed by a line suppressed message.
- The dump page is interpreted on the right-hand side of the printout.

Error Handling

Basic error recovery is available for DASD, tape, and printer devices used as IPL or output devices. In addition, the following information may be of value when the system detects errors:

- The CSW is at location X'40'.
- The I/O address is at location X'BA'.
- 32 bytes of sense data are at location X'2E0'.
- The starting and ending addresses of the CP Trace Table are stored in the PSA at X'7B0' and X'7B4', respectively, in addition to the low storage locations.

Under certain error conditions, storage areas may be overlaid. This could cause fields in SFBLOK and DMPINREC to be incorrect. (For example, fields containing date and time information.)

Appendix D. GCS Control Blocks

This appendix describes the layouts of some GCS control blocks and key fields that are used for identifying problems in a VM/SNA environment. The information that is provided is enough to allow you to display the GCS areas that can be relevant when determining the source of a problem.

This appendix describes the format and layout of:

NUCON GCS Nucleus Constant Area
SIE NUCON Extension
EXTWA External Interrupt Handler Work Area
SVCWA SVC Interrupt Handler Work Area
PGMWA Program Interrupt Work Area.

In all the descriptions, the field lengths are shown in decimal units of bytes; the displacements are shown in hexadecimal.

NUCON - GCS Nucleus Constant Area

Hex Displacement	Name	Decimal Length	Field Description
000	NUCIPPSW	8	Initial Program Loading PSW
008	NUCICCW1	8	Initial Program Loading CCW1
010	NUCICCW2	8	Initial Program Loading CCW2
000	NUCRNPSW	8	Restart New PSW
008	NUCROPSW	8	Restart Old PSW
010	NUCADCVT	4	Address of OS CVT
014	NUCBGCOM	4	Address of BGCOM
018	NUCEOPSW	8	External Old PSW
020	NUCSOPSW	8	SVC Old PSW
028	NUCPOPSW	8	Program-Check Old PSW
030	NUCMOPSW	8	Machine-Check Old PSW
038	NUCIOPSW	8	I/O Old PSW
040	NUCCSW	8	Channel Status Word
048	NUCCAW	4	Channel Address Word
04C	NUCACVT2	4	CVT Address For Dump Routines
050	NUCTIMER	4	Interval Timer
054	NUCTRACE	4	Address of Table Trace Header
058	NUCENPSW	8	External New PSW
060	NUCSNPSW	8	SVC New PSW
068	NUCPNPSW	8	Program-Check New PSW
070	NUCMNPSW	8	Machine-Check New PSW
078	NUCINPSW	8	I/O New PSW
080	NUCSYSCM	4	Used by VSAM
084		2	Reserved - Set to Zero
086	NUCEICOD	2	External Interruption Code
088		1	Reserved - Set to Zero
089	NUCSVILC	1	SVC ILC
08A	NUCSVCN	2	SVC Interruption Code
08C		1	Reserved - Set to Zero
08D	NUCPIILC	1	Program-Check ILC
08E	NUCPICOD	2	Program Interruption Code
090	NUCTEA	1	Reserved - Set to Zero
091	NUCTEAA	3	Translation Exception Address
094		1	Reserved - Set to Zero
095	NUCMCNUM	1	Monitor Call Class Number
096	NUCPERCD	1	Program Event Recorder Code
097		1	Reserved - Set to Zero
098	NUCPER	1	Reserved - Set to Zero
099	NUCPERAD	3	Program Event Recorder Address
09C		1	Reserved - Set to Zero
09D	NUCMTRCD	3	Monitor-Call Code
0A0		8	Reserved for Future Use
0A8	NUCMCKLA	0	Machine-Check Logout Area
0A8	NUCCHNID	4	Channel ID
0AC	NUCIOEL	1	Reserved for Future Use
0AD	NUCIOELA	3	I/O Extended Logout Pointer
0B0	NUCLCL	4	Limited Channel Logout (ECSW)
0B4		4	Reserved for Future Use
0B8		1	Reserved for Future Use
0B9	NUCIOAA	3	I/O Address

Hex Displacement	Name	Decimal Length	Field Description
0BC		44	Reserved for Future Use
0E8	NUCMCIC	8	Machine Check Interruption Code
0F0		8	Reserved for Future Use
0F0		8	Reserved for Future Use
0F8	NUCFSA	1	Reserved - Set to Zero
0F9	NUCFSAA	3	Failing Storage Address
0FC	NUCRGNCD	4	Region Code
100	NUCFLOGA	96	Fixed Logout Area
160	NUCFPRLG	32	Floating Point Register Save Area
180	NUCGPRLG	64	General Purpose Register Save Area
1C0	NUCECRLG	64	Extended Control Register Save Area
200	NUCVTAM	4	Reserved for VTAM
204	NUCVMID	8	Virtual Machine User ID
20C	NUCLVL	4	Release/Service Level
210	NUCIDS	0	SIGNALID/TASKID
210	NUCSIGID	2	This Virtual Machine Signal ID
212	NUCATID	2	Active Task ID
214	NUCATB	4	Address of Active Task
218	NUCPOST	4	Branch Entry Address for POST
21C	NUCCTB	4	Common Trace Block Pointer
220	NUCNPM	4	Network Performance Monitor
224		4	Reserved
228	NUCZIT	4	Start of Private Storage (IPCS Use Only
22C	NUCAGW	4	AGW RAS Use
230		92	Reserved for Future Use
28C	NUCFEIBM	12	Component ID-IPCS Referenced
298	NUCABW	4	Address of Abend Work Area (for IPCS)
29C	NUCRSTS1	4	System Restart Save Area
2A0	NUCRSTS2	4	System Restart Save Area
2A4	NUCRSTF	1	System Restart Flags
2A4	NUCRSTR	X'01'	Recursion Bit (Restart)
2A4	NUCMSGR	X'02'	Recursion Bit (Message Facility)
2A5		3	Reserved
2A8	NUCBLRSV	64	Register Save Area
2E8	NUCCMDLN	160	Command Input Line
388	NUCCMLST	536	Tokenized PLIST
5A0	NUCUPPER	4	Upper Case Translate Table
5A4	NUCPLFID	4	Flag Word Used by CSISCN
5A4	NUCPLSWT	1	1-Byte Switch Used in CSISCN
5A8	NUCCWR	4	Console Write Routine
5AC	NUCACPF	4	CP Command Passthru
5B0	NUCSCANN	4	Scan Routine Entry Point
5B4	NUCSCNT	4	Scan Routine Entry Point
5B8	NUCPLIST	8	Extended PLIST (Untokenized)
5B8	NUCPLCMD	4	Adr of Command Token
5BC	NUCPLBEG	4	Adr of Start of Argument String
5C0	NUCPLEND	4	Adr of End of Argument String
5C4	NUCSIE	4	Pointer to SI Extension
5C8	NUCIHCSA	8	Interrupt Handler common Save Area
5D0	NUCSAVQ1	4	Header Ptr for Interrupt Handler Save Areas
5D4	NUCSAVQ2	4	Trailer Ptr for Interrupt Handler Save Areas
5D8	NUCSRPTR	4	Ptr to System Restart Work Area (for IPCS)

Hex Displacement	Name	Decimal Length	Field Description
5DC	NUCDEB	4	DEB Entry Chain Address
5E0		4	Reserved for Future Use
5E4	NUCCBLKS	4	Pointer to Modules Known to Program Management
5E8		32	Reserved for Future Use
608		72	Reserved for Future Use
650	NUCFCBTB	8	FCB Anchor Chain
650	NUCFCB1	4	Address of First FCB
654	NUCFCBNM	2	Number of FCBs in the Chain
656		2	Reserved
658	NUCLAF	4	V(CSILAF) AACTLKP
65C	NUCERS	4	V(CSIERS) AERASE
660	NUCSTTN	4	V(CSISTT) AESTATE
664	NUCFNS	4	V(CSIFNS) AFINIS
668	NUCFVS	4	V(FVS) AFVS
66C	NUCAUD	4	V(CSIAUD) AUPDISK
670	NUCRDBUF	4	V(CSIRWBRD) CSIRWBRD
674	NUCDEVTB	4	V(DEVTAB) Address of DEVTAB
678	NUCADTS	4	V(ADTSECT) Address of ADTSECT
67C	NUCDIODA	4	V(DIOSECT) Address of DIODA
680	NUCAFTS	4	V(AFTSTART) Address of AFTSTART
684		4	RESERVED
688	NUCTODCA	8	
688	NUCTODTT	8	Total Virtual Machine Time
690	NUCTODDT	8	Time of Day When Dispatched
698		8	Reserved
6A0	NUCLLNAM	4	Address of LOADLIB Name List
6A4	NUCLLDIR	4	Address of LOADLIB Directory List
6A8	NUCLLSIZ	4	Size of LOADLIB Name and Directory Storage
6AC	NUCLLNUM	2	Number of Globaled LOADLIBs
6AE		2	Reserved for Future Use

SIE - NUCON Extension

Hex Displacement	Name	Decimal Length	Field Description
000		8	Eye Catcher (CSISIE)
008	SIETRQ	4	Timer Request Queue Start
00C	SIEQCB	4	ENQ Control Block Queue Start
010	SIETTBL	4	Addr of TASKID Table
014	SIETBQ	4	Addr of 1st Task Block in Dispatch Queue
018	SIEAEQ	4	Addr of Asynchronous Exit Queue
01C	SIESCB	4	Pointer to STAE Control Block Pool
020	SIELKCOM	4	Address of the Common Storage Lock
024	SIELKTID	2	Task ID Waiting for Lock
026	SIELOCKB	1	Byte Indicating Whether the Machine
	SIELKCMB	X'80'	Is Waiting for Lock
027	SIEPM	1	Program Management Flag Byte
	SIEPMGLB	X'80'	Set On When Global LOADLIB Command Issued Set Off When BLDL Searches Directories
	SIEPMOSR	X'40'	Set On When OSRUN is Active

Hex Displacement	Name	Decimal Length	Field Description
			Set Off By Link
028	SIEVMCBS	4	Address of VMCB Array
02C	SIEVMCB	4	Address of This Machine's VMCB
030	SIESYSNM	4	Pointer to VSAM SYSNAMES Table
034	SIEPOST	4	Branch Entry to POST
038	SIEGETM	4	Branch Entry Point to GETMAIN
03C	SIEFREM	4	Branch Entry Point to FREEMAIN
040	SIESMAB	4	Pointer to SMAB
044	SIECAADR	4	Addr of ATTN Interrupt ECB
048	SIECIADR	4	Addr of I/O Complete ECB
04C	SIECOADR	4	Addr of Cons. Output Pend. ECB
	SIECEOL	X'80'	Indicate End of ECB List
050	SIECTADR	4	Addr of Command Task ECB
054	SIECAECB	4	Attention Interrupt ECB
058	SIECIECB	4	I/O Complete ECB
05C	SIECOECB	4	Output Pending ECB
060	SIECTECB	4	Command Task ECB
064	SIECONFL	1	Console Task Flags
	SIECRDIO	X'80'	Read I/O In Progress
	SIECWRIO	X'40'	Write I/O in Progress
	SIECATTP	X'20'	Attention Pending Bit
	SIECOUTP	X'10'	Output Pending Bit
065	SIECMDFL	3	Reserved Command Flags
068	SIEFCMDQ	4	Ptr First Command Input Buf
06C	SIELCMDQ	4	Ptr Last Command Input Buf
070	SIEFSWQE	4	Ptr First WQE Buf on Queue
074	SIELSWQE	4	Ptr Last WQE Buf on Queue
078	SIEFSORE	4	Ptr First ORE BUF on Queue
07C	SIELSORE	4	Ptr Last ORE BUF on Queue
080	SIECCWS	8	Console CCWS
080	SIECCW1	8	First CCW
080	SIECCW1C	1	CCW Command Code
081	SIECCW1A	3	Data Address
084	SIECCW1F	1	Flag Byte
085	SIECCW1N	1	Unused Flag Byte
086	SIECCW1B	2	Byte Count
088	SIECCW2	8	Second CCW
088	SIECCW2C	1	CCW Command Code
089	SIECCW2A	3	Data Address
08C	SIECCW2F	1	Flag Byte
08D	SIECCW2N	1	Unused Flag Byte
08E	SIECCW2B	2	Byte Count
090	SIEIDORE	13	Bit String For ORE IDS
09D	SIELSTID	1	Last ID Used for Assigning
09E		2	Reserved
0A0	SIETAB	4	Trace Anchor Block Pointer
0A4	SIENUCX	4	Pointer to Nucleus Extension Control Block Chain
0A8	SIEBVSAM	4	Beginning of VSAM Shared Segment
0AC	SIEEVSAM	4	End of VSAM Shared Segment
0B0	SIEBBAM	4	Beginning of BAM Shared Segment
0B4	SIEEBAM	4	End of BAM Shared Segment

Hex Displacement	Name	Decimal Length	Field Description
0B8	SIEIUCAB	4	IUCV Anchor Block
0BC	SIESSPTH	2	Signal Services Path (Path ID)
0BE		2	Reserved
0C0	SIEFREST	4	Start of Available Common Free STOR
0C4	SIEZNR	4	Start of Available Private Free STOR
0C8	SIEVMSIZ	4	Size of This Virtual Machine
0CC	SIETQE	4	Address of TQE POOL
0D0		4	Reserved for Future Use
0D4	SIEIFLAG	1	Initialization Flags
	SIEAUSER	X'02'	On Means Virtual Machine Authorized
	SIE4K	X'01'	On Means This is a 4K Page Machine
0D5	SIETIME	8	System Save Time
0DD	SIEDATE	8	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
0E6		2	Reserved
0E8	SIEREDRN	4	Highest Ready Task Level
0EC	SIEDSP	1	
	SIEDSTOP	X'80'	Priority Change Bit
0ED		3	Reserved
0F0	SIESLICE	8	Time Slice in Microseconds
0F8	SIESDXBR	4	Branch Entry to SCHEDEX
0FC	SIESAV	4	Save Area for Branch Entry
100	SIEIUS	4	Branch Entry to IUCV
104	SIEGENIO	4	Branch Entry to GENIO START/R
108	SIESATB	4	Saved Active Tsk Blk Adr
10C	SIESATID	2	Saved Active Tsk Blk ID
10E		2	Reserved

EXTWA - External Interrupt Handler Work Area

Hex Displacement	Name	Decimal Length	Field Description
000	EXTWA	0	External Interrupt Handler Work Area
000	EXTPSW	8	External Old PSW
008	EXTSAVE	80	Save Area
058	EXTAREA	72	Save Area
0A0	EXTREGS	64	Registers at Time of Interrupt
0E0	EXTFPR	32	Floating Point Registers

SVCWA - SVC Interrupt Handler Work Area

Hex Displacement	Name	Decimal Length	Field Description
000	SVCWA	0	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	SVCSTB	176	Default State Block
110	SVCUSA	96	Default User Save Area
170	SVCSTPTR	4	Ptr to State Block in Use
174	Reserved	4	Reserved
178	SVCNQRY	0	PLIST for NUCEXT QUERY
178	SVCNFUNC	8	=CL8'NUCEXT' Identifies NUCEXT Function
180	SVCNNAME	8	=CL8' ' Nucleus Extension Name
188	SVCNPTR	4	Received Ptr to NUCXBLK
18C	SVCNIND	4	=XL4'FFFFFFFF' Identifies NUCEXT QUERY FUNCT

PGMWA - Program Interrupt Work Area

Hex Displacement	Name	Decimal Length	Field Description
000	PGMWA	0	Program Check Interrupt Work Area
000	PGMOPSW	8	Program Old PSW
008	PGMREGS	64	Registers at Time of Interrupt

VMCB - Virtual Machine Control Block

Hex Displacement	Name	Decimal Length	Field Description
000	VMCUSER	8	Virtual Machine Userid
008	VMCINSIG	4	Initialization Signal-id
008		2	Reserved
00A	VMCSIGID	2	Virtual Machine Signal-id
00C	VMCLCKH	4	Lock Holding Pointer
010	VMCLKWD	4	Lock Waiting Pointer
014	VMCSCHDX	4	Pointer to the Chain of AEB Blocks to be Scheduled for this Virtual Machine

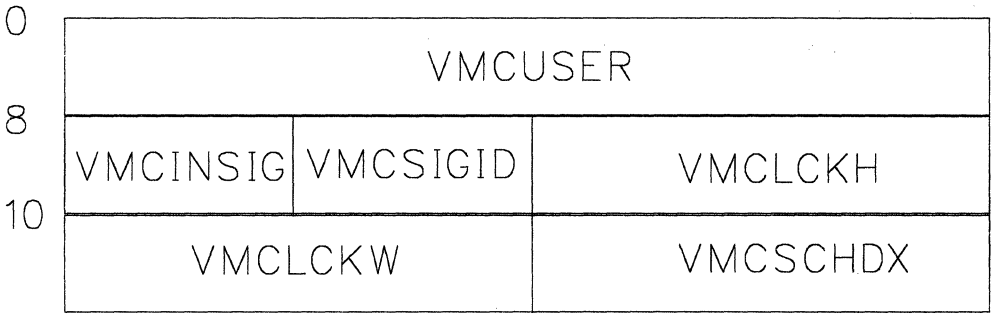


Figure 21. VMCUSER

Summary of Changes

Summary of Changes for VM/SP Diagnosis Guide

To obtain prior editions of *VM/SP Diagnosis Guide* or the *VM/SP System Programmer's Guide*, you must order using a pseudo-number assigned to the respective edition. These pseudo-numbers are found in the *VM/SP Library Guide and Master Index*, GC19-6207.

Summary of Changes
for LY24-5241-1
for VM/SP Release 6

Enhanced CPTRAP Facility

The CPTRAP facility is enhanced with

- The addition of the DATATRAP function which collects various types of data at various selected points in CP.
- The addition of the CCWTRACE function which traces I/O activity between CP and selected real devices.
- The elimination of TRAPRED as the method used to display and analyze records in the CPTRAP file. IPCS now provides support to analyze these records, as well as, formatting and printing records in the CPTRAP file.

New SET AUTODUMP and QUERY AUTODUMP Commands

These new commands support debugging in a remote environment. The SET AUTODUMP command controls the creation of an automatic dump of CMS in the event of an abend. The QUERY AUTODUMP command returns the current setting of the SET AUTODUMP command. Information on these commands starts on page 132.

Shared File System (SFS)

The Shared File System helps CMS manage and share files. Files stored in SFS can be shared by multiple users and across multiple systems. SFS allocates DASD space dynamically as a user creates and extends files; it similarly reclaims the space as users erase or replace files. Users can organize their files in multiple hierarchical structured directories; hence, the same files can be in multiple directories.

Chapter 5, "Debugging the SFS Server Machine" on page 145 has been added to this manual. For more details on SFS, also refer to the *VM/SP CMS User's Guide*, SC19-6210 and *VM/SP CMS Shared File System Administration*, SC24-5367.

APPC/VM VTAM Support (AVS)

VM/SP Advanced Program-to-Program Communications connectivity is enhanced for VM/SP Release 6. A major part of this enhancement is the APPC/VM VTAM Support (AVS) to support the APPC/VM interface to the SNA network.

Chapter 8, "Debugging AVS" on page 235 has been added to this manual. For more details on AVS, also refer to the *VM/SP Connectivity Programming Guide and Reference*, SC24-5377 and *VM/SP Connectivity Planning, Administration, and Operation*, SC24-5378.

Structural Changes

Sections of the VM/SP Problem Reporting Guide, SC24-5282 and the GCS Diagnosis Reference, LY24-5239, have been added to this manual. These two manuals have been discontinued as part of the VM Library.

Because of the enhancement to the IPCS component for Release 6, the *VM/SP Interactive Problem Control System Guide and Reference* has been recreated and IPCS information has been removed from this manual.

Integration of Between-Release Support Information to VM/SP Release 6

The information from the following publications has been added to this book for Release 6:

- *VM VM/VTAM and NetView™ Enhancements*, GC24-5310.
- *VM/SP 9370 Processors, 9332 and 9335 Direct Access Storage Devices, and 9347 Tape Drive*, GC24-5315
- *VM/SP GCS/VSAM Support for Local Shared Resources/Deferred Write*, GC24-5360
- *VM/SP Transparent Access Facility 9370 Local Area Network Subsystems*, GC24-5363

Miscellaneous

Minor technical and editorial changes have been made throughout this manual.

Structural Changes for VM Release 5

This manual contains material formerly found in the *VM/SP System Programmer's Guide* (SC19-6203) or *VM/SP HPO System Programmer's Guide* (SC19-6224), *VM/SP Group Control System Guide* (SC24-5249), and *VM/SP Interactive Problem Control System Guide* (SC24-5260).

Technical Changes for VM Release 5

Summary of Changes
for LY24-5241-0
for VM Release 5

Transparent Services Access Facility (TSAF)

Is a facility that lets users connect to and communicate with local or remote virtual machines within a group of systems. With TSAF, a user can connect to a program by specifying a name that the program has made known, instead of specifying a user ID and node ID.

High Performance Option (HPO)

This manual was updated so that it applies to both VM/SP and VM/SP HPO.

Manual Organization

Chapters 45, 46, and 47 from the *VM/SP System Programmer's Guide* were moved into Chapters 1, 3, and 4 of this manual.

Information on PER and TRACE from the *VM/SP CMS User's Guide* and *VM/SP CP General User Command Reference* were moved into Chapter 2 of this manual.

Information on abend dumps from the *VM/SP Operator's Guide* were moved into Chapter 3 of this manual.

Information on network dump and NCPDUMP from the *VM/SP Operator's Guide* was moved into Chapter 3 of this manual.

Information on the Stand-Alone Dump Facility from the *VM/SP Operator's Guide* was moved into Chapter 3 of this manual.

NetView is a trademark of the International Business Machines Corporation.

Part of Chapter 2 from the *VM/SP Group Control System Guide* was moved into Chapter 5 of this manual.

Chapters 1, 2, and 3 from the *VM/SP Interactive Problem Control System Guide* was moved into Chapter 7 of this manual.

Chapter 4 from the *VM/SP Interactive Problem Control System Guide* and Appendix B from the *VM/SP Group Control System Guide* were moved into Appendix A of this manual.

Appendix B is new to this manual.

Appendix C from the *VM/SP Interactive Problem Control System Guide* was moved into Appendix C of this manual.

Appendix A from the *VM/SP System Programmer's Guide* was moved into Appendix D of this manual.

Information on Stand-Alone Dump Formats from the *VM/SP Operator's Guide* was moved into Appendix E of this manual.

Information on converting symptom summary and dump files from the *VM/SP Interactive Problem Control System Guide* was moved into Appendix F of this manual.

Appendix G is new to this manual.

Miscellaneous

Minor technical and editorial changes have been made throughout this manual.

Some error messages changed to mixed case.

Glossary of Terms and Abbreviations

A

abend. (1) Abnormal end of task. (2) Synonym for *abnormal termination*.

abend dump. The contents of main storage, or part of main storage, written to an external medium for debugging an error condition that resulted in the termination of a task before its regular completion.

abnormal end of task (abend). Termination of a task before its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing.

abnormal termination. The ending of processing before planned termination. Synonymous with *abend*.

accept. Allowing a connection to the user's virtual machine from another virtual machine or from the user's own virtual machine.

ACF/SSP. Advanced Communications Function for Systems Support Programs.

active disk table (ADT). A table residing in the user's copy of the CMS nucleus that contains an entry for each valid file mode letter; that is, for each disk or SFS directory accessed.

active file table (AFT). A table residing in the user's copy of the CMS nucleus that contains an entry for each CMS file currently open.

address stop. See *breakpoint* and *instruction address stop*.

ADT. Active disk table.

Advanced Communications Function for Systems Support Programs (ACF/SSP). An IBM program product made up of a collection of utilities and small programs. SSP is required for operation of the NCP.

Advanced Program-to-Program Communications (APPC). The inter-program communication service within SNA LU 6.2 on which the APPC/VM interface is based.

Advanced Program-to-Program Communications/VM (APPC/VM). An API for communicating between two virtual machines that is mappable to the SNA LU 6.2 APPC interface and based on IUCV functions. Along with the TSAF virtual machine, APPC/VM provides this communication within a single system and throughout a collection of systems.

AFT. Active file table.

alias. A pointer to a base file. An alias can be in the same directory as the base file or in a different directory. There must always be a base file for the alias to point to. The alias references the same data as the base file. Data is not moved or duplicated.

AP. Attached processor.

APAR. Authorized program analysis report.

APAR number. The number that IBM assigns to an APAR and to the change resulting from it.

APPC. Advanced Program-to-Program Communications.

APPC/VM. Advanced Program-to-Program Communications/VM.

APPC/VM VTAM Support (AVS). A component of VM/SP that lets application programs using APPC/VM communicate with programs anywhere in a network defined by IBM's SNA. AVS transforms APPC/VM into APPC/VTAM protocol.

application program. A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll.

apply. When servicing a product or component, to generate an auxiliary control structure from a PTF.

area. A term acceptable for DASD space when there is no need to differentiate between space on count-key-data devices and FB-512 devices. See *DASD space*.

assembler language. A source language that includes symbolic machine language statements in which there is a one-to-one correspondence with instruction formats and data formats of the computer.

attached processor (AP). A processor that has no I/O capability and is always linked to the processor initialized for I/O handling.

attention interrupt. An I/O interrupt caused by a terminal user pressing the attention key (or equivalent). See *attention key (ATTN key)* and *signaling attention*.

attention key (ATTN key). A function key on terminals that, when pressed, causes an I/O interruption in the processing unit. See *signaling attention*.

ATTN key. Attention key.

authority. In SFS, the permission to access a file or directory. You can have read authority or write authority (which includes read authority). You can also have file pool administration authority, which is the highest level of authority in a file pool.

authorized program. Synonym for *privileged program.*

authorized program analysis report (APAR). An official request to the responsible IBM Change Team to look into a suspected problem with IBM code or documentation. APARs describe problems giving conditions of failure, error messages, abend codes, or other identifiers. They also contain a problem summary and resolution when applicable. See *program temporary fix (PTF).*

authorized user ID. In GCS, a user ID that provides access to the GCS supervisor, supervisor state, and (in some cases) certain restricted CP commands. This access is provided by including the user ID on a list of authorized user IDs compiled with the GCS GROUP EXEC. The virtual machine associated with an authorized user ID is an *authorized* machine, and programs running in that machine are *authorized* applications.

authorized virtual machine. A GCS virtual machine identified by user ID.

auxiliary directory. In CMS, an extension of the CMS file directory for a minidisk, which contains the names and locations of certain CMS modules not included in the minidisk's CMS file directory.

AVS. APPC/VM VTAM Support.

AVS virtual machine. The virtual machine that manages a gateway that allows communication between VM systems and an SNA network.

B

basic control (BC) mode. A mode in which additional System/370 features, such as new machine instructions, are not operational. Contrast with *extended control (EC) mode.*

BC mode. Basic control mode.

binary digit. Either of the digits 0 or 1 when used in the pure binary numeration system. Synonymous with *bit.*

binary synchronous communication (BSC). Communication using binary synchronous line discipline in which transmission of binary-coded data between stations is synchronized by timing signals generated at the sending and receiving stations.

bit. (1) Either of the binary digits 0 or 1. See *byte.* (2) Synonym for *binary digit.*

block. A unit of DASD space on FB-512 devices. For example, FB-512 devices can be the IBM 9335, 9332, 9313, 3370, and 3310 DASD using fixed-block architecture.

bpi. Bits per inch.

Bpi. Bytes per inch.

breakpoint. A place in a program, specified by a command or a condition, where the system halts execution and gives control to the workstation user or to a specified user.

BSC. Binary synchronous communication.

buffer. An area of storage, temporarily reserved for performing input or output, into which data is read, or from which data is written.

build. In reference to installation and service of a product, to do the necessary steps to produce executable code or systems. This is often called the *build process.*

byte. A unit of storage, consisting of eight adjacent binary digits that are operated on as a unit and constitute the smallest addressable unit in the system.

C

CAW. Channel address word.

CC. Condition code.

CCS. Console communication service.

CCW. Channel command word.

changes. In reference to installation and service, IBM and original equipment manufacturer (OEM) supplied service for their programs. In the IBM service process, there are many ways users can receive information they need to fix (change) a portion(s) of a product they are running on a VM system. These include PTFs, APARs, user modifications, and information received over the phone. All these types of information are called *changes.*

channel. A path in a system that connects a processor and main storage with an I/O device.

channel address word (CAW). An area in storage that specifies the location in main storage at which a channel program begins.

channel-check handler (CCH). In System/370, a feature that records information about channel errors and issues appropriate messages to the operator.

channel command word (CCW). A doubleword at the location in main storage specified by the channel address word. One or more CCWs make up the channel program that directs data channel operations.

channel status word (CSW). An area in storage that provides information about the termination of I/O.

channel-to-channel adapter (CTCA). A hardware device that connects two channels on the same computing system or on different systems.

checkpoint. An internal file pool server operation during which the changes recorded on the log minidisks are permanently made to the file pool.

checkpoint (CKPT) start. A VM/SP system restart that attempts to recover information about closed spool files previously stored on the checkpoint cylinders. The spool file chains are reconstructed, but the original sequence of spool files is lost. Unlike warm start, CP accounting and system message information is also lost. Contrast with *cold start, force start,* and *warm start.*

CICS/VM. Customer Information Control System for VM.

circumventive service. Information that IBM supplies over the phone or on a tape to circumvent a problem by disabling a failing function until a PTF is available to be shipped as a corrective service fix. See *patch* and *zap.*

CKD. Count-key-data.

class authority. Privilege assigned to a virtual machine user in the user's directory entry; each class specified allows access to a subset of all the CP commands. See *privilege class* and *user class restructure (UCR).*

class C user. See *system programmer privilege class.*

clock comparator. A hardware feature (required by VM/SP) that causes an interruption when the TOD clock has equaled or exceeded the value specified by a program or virtual machine.

CMS. Conversational Monitor System.

CMS/DOS. The functions of CMS that become available when the user enters the command: SET DOS ON. CMS/DOS is a part of the regular CMS system and is not a separate system. Users who do not use CMS/DOS are sometimes called OS users, because they use the OS simulation functions of CMS. Synonymous with *DOS simulation under CMS.* Contrast with *OS simulation under CMS.*

CMS EXEC. An EXEC procedure or EDIT macro written in the CMS EXEC language and processed by the CMS EXEC processor. Synonymous with *CMS program.*

CMS EXEC language. A general-purpose, high-level programming language, particularly suitable for EXEC procedures and EDIT macros. The CMS EXEC processor executes procedures and macros (programs) written in this language. Contrast with *EXEC 2 language* and *Restructured Extended Executor (REXX) language.*

CMS file directory. A directory on each CMS disk that contains the name, format, size, and location of each of the CMS files on that disk. When a disk is accessed by the ACCESS command, its directory is read into virtual storage and identified with any letter from A through Z. Synonymous with *master file directory block* and *minidisk directory.*

CMS nucleus. The portion of CMS that is resident in the user's virtual storage whenever CMS is executing. Each CMS user receives a copy of the CMS nucleus when the user IPLs CMS. See *saved system* and *shared segment.*

CMS program. Synonym for *CMS EXEC.*

cold start. A VM/SP system restart that ignores previous data areas and accounting information in main storage, and the contents of paging and spool files on CP-owned disks. Contrast with *checkpoint (CKPT) start, force start,* and *warm start.*

collection. See *TSAF collection.*

command. A request from a user at a terminal for the execution of a particular CP, CMS, IPCS, GCS, TSAF, or AVS function. A CMS command can also be the name of a CMS file with a file type of EXEC or MODULE. See *subcommand* and *user-written CMS command.*

command line. The line at the bottom of display panels that lets a user enter commands or panel selections. It is prefixed by an arrow (= = = >).

common dump receiver. One user ID in a virtual machine group appointed to receive other group members' storage dumps. Unless the user specifies otherwise, all dumped information automatically goes to this user ID (identified with the GCS GROUP EXEC). It should be an authorized user ID in order to receive fetch-protected data as well as storage with a key other than 14.

common lock. A doubleword in storage, controlled by the GCS LOCKWD macro. When a program is using common storage, it can turn the common lock ON.

Other programs that examine the lock and find it ON cannot gain access to common storage.

common storage. A shared segment of reentrant code that contains free storage space, the GCS supervisor, control blocks, and data that all members of a virtual machine group share.

communication link. Synonym for *data link*.

compile. To translate a program written in a high-level programming language into a machine language program.

component. A collection of objects that together form a separate functional unit. A product may contain many components (for example, VM/SP has components of CP, CMS, GCS, TSAF, IPCS, and AVS). A component can be part of many products. (CP spans both VM/SP and VM/HPO products).

component override. Synonym for *component parameter override*.

component parameter override. A component parameter, defined in a component override area, that updates or replaces a component parameter defined in a component area of the product parameter file. Synonymous with *component override* and *override*.

concurrently. Concerning a mode of operation that includes doing work on two or more activities within a given (short) interval of time.

condition code (CC). A code that reflects the result of a previous I/O, arithmetic, or logical operation.

connect. Establishing a path to communicate with another virtual machine or with the user's own virtual machine.

console. A device used for communications between the operator or maintenance engineer and the computer.

console communication service (CCS). A group of CP modules that interfaces with the VTAM service machine, providing full VM/SP console capabilities for SNA terminal users.

console function. The subset of CP commands that lets the user simulate almost all of the functions available to an operator at a real system console.

console spooling. Synonym for *virtual console spooling*.

console stack. Refers collectively to the program stack and the terminal input buffer.

contention. The situation where two LUs try to allocate a conversation over the same session at the same time.

control block. A storage area that a computer program uses to hold control information.

control data. In reference to a file pool, the data that controls the DASD space and objects within a file pool. Control data consists of the POOLDEF file, the control minidisk, and all minidisks allocated to storage group 1.

control file. (1) In service, a file with file type CNTRL that contains records that identify the updates to be applied and the macro libraries, if any, needed to assemble that source program. (2) A CMS file that is interpreted and directs the flow of a certain process through specific steps. For example, the control file could contain installation steps, default addresses, and PTF prerequisite lists as well as many other necessary items.

control program. A computer program that schedules and supervises the program execution in a computer system. See *Control Program (CP)*.

Control Program (CP). A component of VM/SP that manages the resources of a single computer so multiple computing systems appear to exist. Each virtual machine is the functional equivalent of an IBM System/370.

control section (CSECT). The part of a program specified by the programmer to be a relocatable unit, all elements of which are loaded into adjoining main storage.

control unit. A device that controls I/O operations at one or more devices.

conversation. A connection between two transaction programs over an LU-LU session that lets them communicate with each other while processing some transaction. The programs establish a conversation, send and receive data in the conversation, and then terminate the conversation.

Conversational Monitor System (CMS). A virtual machine operating system and component of VM/SP that provides general interactive time sharing, problem solving, program development capabilities, and operates only under the control of the VM Control Program (CP).

corrective service. Service that IBM supplies on tape to correct a specific problem.

count-key-data (CKD) device. A disk storage device that stores data in the format: count field, usually followed by a key field, followed by the actual data of a record. The count field contains the cylinder number, head number, record number, and the length of the data. The key field contains the record's key (search argument).

CP. Control Program.

CP assist. A hardware function, available only on a processor with ECPS, that reduces CP overhead by doing the most frequently used tasks of CP routines.

CP command. A command available to all VM users. Class G CP commands let the general user reconfigure their virtual machine, control devices attached to their virtual machine, do input and output spooling functions, and simulate many other functions of a real computer console. Other CP commands let system operators, system programmers, system analysts, and service representatives manage the resources of the system.

CP directory. Synonym for *VM/SP directory*.

CP trace table. A table VM/SP uses for debugging. Its size is a multiple of 4096 bytes and depends on the size of real storage or a user specified value. This table contains the chronological occurrences of events that take place in the real machine, recorded in a wraparound fashion within the trace table. Synonymous with *trace table*.

CPTRAP. A CP debugging tool that creates a reader spool file of selected trace table entries, CP data, and virtual machine data in the order that they happen. The IPCS commands can help the user access and print this collected data.

CPU timer. A hardware feature that measures elapsed processor time and causes an interruption when a previously specified amount of time has elapsed. The CPU timer is decremented when the processor is executing instructions, is in a WAIT state, and is executing program loading instructions, but not when the processor is in a stopped state. A virtual machine that uses the CPU timer must have the EC mode and REALTIMER options active.

CSECT. Control section.

CSW. Channel status word.

CTCA. Channel-to-channel adapter.

Customer Information Control System for VM (CICS/VM). An IBM licensed program that provides a transaction processing capability for use in distributed departmental VM systems. It lets users in individual departments mix transaction processing requests with other office and commercial applications.

CVT. Communications vector table.

cylinder. In a disk pack, the set of all tracks with the same nominal distance from the axis about which the disk pack rotates.

D

DASD. Direct access storage device.

DASD Dump Restore (DDR) program. A service program that copies all or part of a minidisk onto tape, loads the contents of a tape onto a minidisk, or sends data from a DASD or from tape to the virtual printer.

DASD space. (1) Area allocated to DASD units on CKD devices. (2) Area allocated to DASD units on FB-512 devices. Note that *DASD space* is synonymous with *cylinder* when there is no need to differentiate between CKD devices and FB-512 devices. This term applies to VM/370, VM/SP and VM/SP HPO program products.

DAT. Dynamic address translation.

data link. The equipment and rules (protocols) used for sending and receiving data. Synonymous with *communication link*.

DDR program. DASD Dump Restore program.

dedicated device. An I/O device or line not being shared among users. The facility can be permanently assigned to a particular virtual machine by a VM/SP directory entry, or temporarily attached by the resource operator to the user's virtual machine.

default operand. An operand that has a preset value if a value is not specified on the CP or CMS command line.

direct access storage device (DASD). A storage device in which the access time is effectively independent of the location of the data.

directory. See *auxiliary directory, CMS file directory, SFS directory,* or *VM/SP directory*.

discontiguous saved segment. One or more 64K segments of storage that were previously loaded, saved, and assigned a unique name. The segment(s) can be shared among virtual machines if the segment(s) contains reentrant code. Discontiguous segments used with CMS must be loaded into storage at locations above the address space of a user's CMS virtual machine. They can be detached when no longer needed.

disk. A magnetic disk unit in the user's CMS virtual machine configuration. Also called a virtual disk.

disk operating system (DOS). An operating system for computer systems that use disks and diskettes for auxiliary storage of programs and data.

dispatcher. The program in CP that places virtual machines or CP tasks into execution. The dispatcher

selects the next virtual machine to run and prepares the virtual machine for problem state execution.

dispatching. The starting of virtual machine execution.

display mode. A type of editing at a display terminal in which an entire screen of data is displayed at once and in which the user can access data through commands or by using a cursor. Contrast with *line mode*.

display terminal. A terminal with a component that can display information on a viewing surface such as a CRT or gas panel.

DOS. Disk operating system.

DOS simulation under CMS. Synonym for *CMS/DOS*.

dump. To write the contents of part or all of main storage, or part or all of a minidisk, to auxiliary storage or a printer. See *abend dump*.

dynamic address translation (DAT). In System/370 virtual storage systems, the change of a virtual storage address to a real storage address during execution of an instruction.

E

EBCDIC. Extended binary-coded decimal interchange code.

ECB. Event control block.

EC mode. Extended control mode.

ECPS:VM/370. Extended Control Program Support:VM/370.

ECSW. Extended channel status word.

edit. A function that makes changes, additions, or deletions to a file on a disk. These changes are interactively made. The edit function also generates information in a file that did not previously exist.

emulation program (EP). A control program that lets an IBM 3704 or 3705 Communications Controller emulate the functions of an IBM 2701 Data Adapter Unit, an IBM 2702 Transmission Control Unit, or an IBM 2703 Transmission Control Unit.

entry point. An address or label of an instruction performed upon entering a computer program, a routine, or a subroutine. A program can have several different entry points, each corresponding to a different function or purpose.

EP. Emulation program.

event control block (ECB). A control block that represents the status of an event.

EXEC 2 language. A general-purpose, high-level programming language, particularly suitable for EXEC procedures and XEDIT macros. The EXEC 2 processor runs procedures and XEDIT macros (programs) written in this language. Contrast with *CMS EXEC language* and *Restructured Extended Executor (REXX) language*.

EXEC procedure. (1) A procedure defined by a frequently used sequence of CMS and CP commands to do a commonly required function. A user creates the procedure to save repetitious rekeying of the sequence, and invokes the entire procedure by entering a command (that is, the exec file's file name). The procedure could consist of a long sequence of CMS and CP commands, along with REXX, EXEC 2, or CMS EXEC control statements to control processing within the procedure. (2) A CMS file with a file type of EXEC.

expanded virtual machine assist. A hardware assist function, available only on a processor that has ECPS, that handles many privileged instructions not handled by VMA, and extends the level of support of certain privileged instructions beyond that provided by VMA.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 characters, with each character represented by 8 bits.

extended control (EC) mode. A mode in which all features of a System/370 computing system, including dynamic address translation, are operational. Contrast with *basic control (BC) mode*.

Extended Control Program Support (ECPS:VM/370). A hardware assist feature that improves the performance of CP by reducing CP overhead. ECPS:VM/370 consists of CP assist, expanded virtual machine assist, and virtual interval timer assist.

extended PLIST (untokenized parameter list). Four addresses that indicate the extended form of a command as it was entered at a terminal.

F

FCB. (1) Forms control buffer. (2) Function control block.

fetch protection. A storage protection feature that determines right-of-access to main storage by matching the protection key associated with a main storage fetch reference with the storage keys associated with those frames of main storage.

FIFO (first-in-first-out). A queuing technique in which the next item to be retrieved is the item that has been

on the queue for the longest time. Contrast with *LIFO (last-in-first-out)*.

file access mode. A file mode number that designates whether the file can be used as a read-only or read/write file by a user. See *file mode*.

file definition. (1) Equating a CMS file identifier (file name, file type, file mode) with an OS data set name by the FILEDEF command; or equating a DOS file ID with a CMS file identifier by the DLBL command. (2) Identifying the input or output files used during execution of a program (by way of either the FILEDEF or DLBL commands).

file ID. A CMS file identifier that consists of a file name, file type, and file mode. The file ID is associated with a particular file when the file is created, defined, or renamed under CMS. See *file name, file type,* and *file mode*.

file mode. A two-character CMS file identifier field comprised of the file mode letter (A through Z) followed by the file mode number (0 through 6). The file mode letter indicates the minidisk or SFS directory on which the file resides. The file mode number indicates the access mode of the file. See *file access mode*.

file name. A one-to-eight character alphanumeric field, comprised of A through Z, 0 through 9, and special characters $ # @ + - (hyphen) : (colon) _ (underscore), that is part of the CMS file identifier and serves to identify the file for the user.

file pool. A collection of minidisks managed by SFS. It contains user files and directories and associated control information. Many users' files and directories can be contained in a single file pool.

file type. A one-to-eight character alphanumeric field, comprised of A through Z, 0 through 9, and special characters $ # @ + - (hyphen) : (colon) _ (underscore), that is used as a descriptor or as a qualifier of the file name field in the CMS file identifier. See *reserved file types*.

first-level storage. Refers to real main storage. Contrast with *second-level storage* and *third-level storage*.

flush list. A set of pages available to replenish the free list.

force start. A VM/SP system restart that attempts to recover information about closed spool files previously stored on the checkpoint cylinders. All unreadable or invalid spool file information is ignored. Contrast with *checkpoint (CKPT) start, cold start,* and *warm start*.

forms control buffer (FCB). In the 3800 Printing Subsystem, a buffer for controlling the vertical format of printed output. The FCB is analogous to the punched-paper, carriage-control tape that IBM 1403 Printers use.

free storage. Storage not allocated. The blocks of memory available for temporary use by programs or by the system.

function control block (FCB). In Subsystem Support Services (SSS), a control block that contains information such as a function's status, event control block, task I/O queue, and I/O queue.

G

gateway. The LU name of a TSAF collection that is a source for communications to an SNA-defined network or the target of communications from an SNA-defined network.

GCS. Group Control System.

general register. In CMS, a register that does operations such as binary addition, subtraction, multiplication, and division. General registers primarily compute and modify addresses in a program.

GPR. General-purpose register.

group. Synonym for *virtual machine group*.

group configuration file. A file that the GROUP EXEC creates. It contains the *blueprint* for building the user's virtual machine group. The name of the file is *systemname* GROUP, where *systemname* is the name of the user's GCS saved system.

Group Control System (GCS). A component of VM/SP, consisting of a shared segment that the user can IPL and run in a virtual machine. It provides simulated MVS services and unique supervisor services to help support a native SNA network.

guest operating system (GOS). A second operating system that runs on the user's primary operating system. An example of a GOS is VSE running on VM/SP to support VM/VCNA.

guest virtual machine (GVM). A virtual machine in which an operating machine is running.

I

immediate command. A type of CMS command that, when entered after an attention interruption, causes program execution, tracing, or terminal display to stop. Another immediate command can be entered to resume tracing or terminal display. The immediate commands are HB (halt batch execution), HI (halt all System Product Interpreter or EXEC 2 programs or macros), HO (halt tracing), HT (halt typing), HX (halt execution), RO (resume tracing), RT (resume typing), SO (suspend tracing), TE (trace end), and TS (trace start). They are called immediate commands because they are executed as soon as they are entered; they are not stacked in the console stack. Within an exec, immediate commands can be established or cancelled by the CMS command IMMCMD.

indicator. A 1-byte area of storage that contains either the character "1" to denote a true condition or the character "0" to denote a false condition.

initial program load (IPL). The initialization procedure that causes an operating system to begin operation. A VM user must IPL the specific operating system into the virtual machine that will control the user's work. Each virtual machine can be loaded with a different operating system.

initialize. To set counters, switches, addresses, or contents of storage to starting values.

input line. For typewriter terminals, information keyed in by a user between the time the typing element of the terminal comes to rest following a carriage return until another carriage return is typed. For display terminals, the data keyed into the user input area of the screen. See *user input area.*

input/output (I/O). (1) Pertaining to a device whose parts can do an input process and an output process at the same time. (2) Pertaining to a functional unit or channel involved in an input process, output process, or both, concurrently or not, and to the data involved in such a process.

instruction address stop. An instruction address specified by a CP or CMS command, which, when fetched, causes the virtual machine to stop.

interactive. The classification given to a virtual machine depending on this virtual machine's processing characteristics. When a virtual machine uses less than its allocation time slice because of terminal I/O, the virtual machine is classified as being interactive. Contrast with *noninteractive.*

Interactive Problem Control System (IPCS). A component of VM/SP that permits online problem management, interactive problem diagnosis, online debugging for disk related CP or virtual machine abend dumps or CPTRAP files, problem tracking, and problem reporting.

interface. A shared boundary between two or more entities. An interface might be a hardware or software component that links two devices or programs together.

internal trace table. See *CP trace table.*

interrupt. A suspension of a process, such as execution of a computer program, caused by an external event and done in such a way that the process can be resumed.

Inter-User Communications Vehicle (IUCV). A VM/SP generalized CP interface that helps the transfer of messages either among virtual machines or between CP and a virtual machine.

invoke. To start a command, procedure, or program.

I/O. Input/output.

IPCS. Interactive Problem Control System.

IPL. Initial program load.

IPL processor. In an AP or MP system, the processor on which the control program was first initialized during system generation. Note that both the IPL and the non-IPL processors in a real MP configuration have I/O capabilities.

IUCV. Inter-User Communications Vehicle.

K

K. kilobyte.

kilobyte (K). 1024 bytes.

L

LIFO (last-in-first-out). A queuing technique in which the next item to be retrieved is the item most recently placed in the queue. Contrast with *FIFO (first-in-first-out).*

line mode. The mode of operation of a display terminal that is equivalent to using a typewriter-like terminal. Contrast with *display mode.*

link. (1) In RSCS, a connection, or ability to communicate, between two adjacent nodes in a network. (2) In TSAF, the physical connection between two systems.

load. In reference to installation and service, to move files from tape to disk, auxiliary storage to main

storage, or minidisks to virtual storage within a virtual machine.

loader. A routine, commonly a computer program, that reads data into main storage.

load map. A map containing the storage addresses of control sections and entry points of a program loaded into storage.

local. Two entities (for example, a user and a server) are said to be local to each other if they belong to the same system within a collection or to the same node within an SNA system. Contrast with *remote*.

local area network (LAN). A data network located on the user's premises in which serial transmission is used for direct data communication among data stations. Contrast with *wide area network (WAN)*.

local service. Changes manually applied to a product or component (that is, not using the program update service or corrective service procedures). See *circumventive service* and *user modification*.

lock. A tool for controlling concurrent usage of SFS objects. Implicit locks are acquired and automatically released when you run CMS commands and program functions in SFS. Explicit locks let you control the type and duration of the lock.

locked page. A page that is not to be paged out.

logical line. A command or data line that can be separated from one or more additional command or data lines on the same input line by a logical line end symbol.

logical unit (LU). An entity addressable within an SNA-defined network, similar to a node within a VM network. LUs are categorized by the types of communication they support. A TSAF collection in an SNA network is viewed as one or more LUs.

logoff. The procedure by which a user ends a terminal session.

logon. The procedure by which a user begins a terminal session.

LU. Logical unit.

M

machine. A synonym for a virtual machine running under the control of VM/370 or VM/SP.

machine ID. A 2-byte field that uniquely defines a virtual machine within a virtual machine group. Machine ID is sometimes combined with task ID to uniquely identify a task within the virtual machine group.

macro. Synonym for *macrodefinition* and *macroinstruction*.

macrodefinition. A set of statements that defines the name of, format of, and conditions for generating a sequence of assembler language statements from a single source statement. Synonymous with *macro*.

macroinstruction. In assembler language programming, an assembler language statement that causes the assembler to process a predefined set of statements called a macrodefinition. The statements usually produced from the macrodefinition replace the macroinstruction in the program. Synonymous with *macro*.

map. In CMS, the file that contains a CMS output listing, such as (1) a list of macros in the MACLIB library, including macro size and location within the library, (2) a listing of the directory entries for the DOS/VS system or private source, relocatable, or core image libraries, (3) a linkage editor map for CMS/DOS programs, and (4) a module map containing entry point locations.

master file directory. A directory on each CMS disk that contains the name, format, size, and location of all the CMS files on the disk. When a disk is accessed by the ACCESS command, the directory is read into main storage and identified with one of the 26 disk mode letters (A through Z).

master file directory block. Synonym for *CMS file directory*.

MB. Megabyte.

megabyte (MB). 1,048,576 bytes.

message. Data sent from a source application to a target application program in a conversation.

minidisk directory. Synonym for *CMS file directory*.

module. (1) A unit of a software product that is discretely and separately identifiable with respect to modifying, compiling, and merging with other units, or with respect to loading and execution. For example, the input to, or output from, a compiler, the assembler, the linkage editor, or an exec routine. (2) A nonrelocatable file whose external references have been resolved.

MP. Multiprocessor.

Multiple Virtual Storage (MVS). An alternative name for OS/VS2.

multiprocessor (MP). A computer using two or more processing units under integrated control.

multitasking. Providing services for many tasks that are active at the same time.

MVS. Multiple Virtual Storage.

N

native mode. Refers to running an operating system stand-alone on the real machine instead of under VM/SP.

NCP. Network control program.

NCPDUMP. Network control program DUMP.

network. Any set of two or more computers, workstations, or printers linked in such a way as to let data be transmitted between them.

network control program (NCP). An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability.

node. (1) A single processor or a group of processors in a teleprocessing network. (2) A computer, workstation, or printer, when it is participating in a network.

node ID. Node identifier.

node identifier (node ID). The name by which a node is known to all other nodes in a network.

noninteractive. The classification given to a virtual machine depending on this virtual machine's processing characteristics. When a virtual machine usually uses all its allocated time slice, it is classified as being noninteractive or compute bound. Contrast with *interactive*.

non-IPL processor. In an AP or MP system, the attached or second processor initialized at system generation time. Note that both the IPL processor and the non-IPL processor in a real MP configuration have I/O capabilities.

nonpaging mode. Synonym for *OS/VS1 nonpaging mode*.

nonprivileged program. In GCS, a program called by a GCS application that operates in problem state. Contrast with *privileged program*.

nucleus. The part of CP, CMS, and GCS resident in main storage.

NUCON. The nucleus constant area of CMS.

O

object code. Compiler or assembler output that is executable machine code or is suitable for more processing to produce executable machine code. Contrast with *source code*.

operand. Information entered with a command name to define the data on which a command processor operates and to control the execution of the command processor.

OS simulation under CMS. The environment of CMS that permits the simulation of OS functions. Contrast with *CMS/DOS*.

OS/VS1. A virtual storage operating system that is an extension of OS/MFT. Synonymous with *nonpaging mode*.

OS/VS2. A virtual storage operating system that is an extension of OS/MVT.

OS/VS1 nonpaging mode. If OS/VS1 executes under the control of a VM/SP system that supports the VM/VS handshaking feature and if the OS/VS1 address space is equal to the size of its VM/SP virtual machine, OS/VS1 executes in nonpaging mode. When OS/VS1 executes in nonpaging mode, it uses fewer privileged instructions and avoids duplicate paging because paging is done only by CP.

overlay. The technique of repeatedly using the same areas of internal storage during different stages of a program.

override. Synonym for *component parameter override*.

P

page. A fixed-length block that has a virtual address and can be transferred between real storage and auxiliary storage.

page frame. A block of 4096 bytes of real storage that holds a page of virtual storage.

page number. The part of a virtual storage address needed to refer to a page.

page table. A table (labeled PAGTABLE) that indicates whether or not a page is in real storage and that correlates virtual addresses with real storage addresses.

page zero. Storage locations 0 to 4095.

paging. Transferring pages between real storage and external page storage.

parameter. A variable that is given a constant value for a specified application and that may denote the application.

parameter list (PLIST). In CMS, a string of 8-byte arguments that call a CMS command or function. The first argument must be the name of the command or function to be called. General register 1 points to the beginning of the parameter list.

part. A CMS file provided on a product tape or service tape as input to the build process. See *build*. A part is the smallest serviceable unit of a component.

patch. A circumventive service change applied directly to object code in a text deck in a nucleus.

path. In APPC/VM or IUCV, a connection between two application programs that are on the same or different systems. Paths have names assigned to them.

performance option. One or more functions that can be assigned to a virtual machine to improve its performance, response time (if terminal-oriented) or throughput under VM/SP.

physical screen. Synonym for *screen*.

PIE. Program interrupt element.

PLIST. Parameter list.

PMA. Preferred machine assist.

preferred machine assist (PMA). The hardware feature of the IBM 308X processor complex or the IBM 3033 processor that improves MVS/SP (Release 1 enhancement, or later) V = R virtual machine performance. The MVS/SP guest virtual machine operates in supervisor state with direct control of its own I/O operations under VM/SP HPO. PMA is an extension of VMA, which eliminates CP simulation of certain instructions and interruptions.

prefix storage area (PSA). A page zero of real storage that contains machine-used data areas and CP global data.

preventive service. The massive application of PTFs from the PUT. Contrast with *selective preventive service*.

private storage. A combination of application code and GCS code available to only one particular virtual machine. No virtual machine can access or share another's private storage area.

privilege class. One or more classes assigned to a virtual machine user in a VM/SP directory entry; each privilege class specified lets a user access a logical subset of the CP commands. There are eight IBM-defined privilege classes that correspond to specific administrative functions. They are:

Class A - primary system operator
Class B - system resource operator
Class C - system programmer
Class D - spooling operator
Class E - system analyst
Class F - service representative
Class G - general user
Class H - reserved for IBM use
Class Any - available to any user.

The privilege classes can be changed to meet the needs of an installation. See *class authority* and *user class restructure (UCR)*.

privileged instruction simulation. The CP-incurred overhead to handle privileged instructions for virtual machine operating systems that execute as if they were in supervisor state but which are executing in problem state under VM/SP. See *virtual machine assist (VMA)*.

privileged program. In GCS, a program called by a GCS application that operates in supervisor state and uses privileged functions. A privileged program is one that meets either of the following requirements:

- It runs in an authorized virtual machine.
- It is called through the AUTHCALL facility.

Synonymous with *authorized program*. Contrast with *nonprivileged program*.

problem state. A state during which the central processing unit cannot execute I/O and other privileged instructions. VM/SP runs all virtual machines in problem state. See *privileged instruction simulation*. Contrast with *supervisor state*.

process. A systematic sequence of operations to produce a specified result. A process is usually logical, not physical.

product. Any separately installable software program, whether supplied by IBM or otherwise, distinct from others and recognizable by a unique identification code. The product identification code is unique to a given product, but does not identify the release level of that product.

PROFILE EXEC. A special EXEC procedure with a file name of PROFILE that a user can create. The procedure is usually executed immediately after CMS is loaded into a virtual machine (also known as IPL CMS).

program stack. Temporary storage for lines (or files) being exchanged by programs that execute under CMS. See *console stack*.

program status word (PSW). An area in storage that indicates the order in which instructions are executed, and to hold and indicate the status of the computer system.

program temporary fix (PTF). Code changes needed to correct a problem reported in an APAR. The corrected code is included in later releases. A PTF contains one or more APAR fixes. For object maintained parts that are changed, the PTF includes replacement parts. For source maintained parts that are changed, the PTF includes update files and replacement parts. Each PTF is unique to a given release of a product. If the same problem occurs in multiple releases of a product, a separate PTF is defined for each release.

program update service. Receiving the contents of a PUT, applying all or some of the changes, and rebuilding the serviced parts. See *preventive service* and *selective preventive service*.

program update tape (PUT). A tape containing a customized collection of service tapes (preventive service) to match the products listed in a customer's ISD (IBM Software Distribution) profile. Each PUT contains cumulative service for the customer's products back to earlier release levels of the product still supported. The tape is distributed to authorized customers of the products at scheduled intervals or on request.

prompt. A displayed message that describes required input or gives operational information.

prompting. An interactive technique that lets the program guide the user in supplying information to a program. The program types or displays a request, question, message, or number, and the user enters the desired response. The process is repeated until all the necessary information is supplied.

PSA. Prefix storage area.

PSW. Program status word.

PTF. Program temporary fix.

PUT. Program update tape.

PVM. VM/Pass-Through Facility.

Q

queue-drop. The action by the system scheduler, DMKSCH, of removing a virtual machine from the list of virtual machines that can be given control of a processor.

R

raddr. The real device address of an I/O device.

read/write access. An access mode associated with a virtual disk or SFS directory that lets a user read and write any file on the disk or SFS directory.

real address. The address of a location in real storage or the address of a real I/O device.

real machine. The actual processor, channels, storage, and I/O devices required for VM/SP operation.

receive. (1) Bringing into the specified buffer data sent to the user's virtual machine from another virtual machine or from the user's own virtual machine. (2) To load service files from a service tape.

recovery machine. The first machine to join a virtual machine group. It has responsibility for executing routines that were set with the GCS MACHEXT macro and cleaning up system resources when machines leave the group.

register. See *general register*.

remote. Two entities (for example, a user and a server) are said to be remote to each other if they belong to different systems within a collection, or to different nodes within an SNA network. Contrast with *local*.

Remote Spooling Communications Subsystem Networking (RSCS). An IBM licensed program and special-purpose subsystem that supports the reception and transmission of messages, files, commands, and jobs over a computer network.

reply. (1) A response to an inquiry. (2) In SNA, a request unit sent only in reaction to a received request unit.

requester. (1) The name given to a virtual machine containing a user program that requests a resource. (2) The program that relays a request to another computer through the SRPI. Contrast with *server*.

reserved file types. (1) File types recognized by the CMS editors (EDIT and XEDIT) as having specific default attributes that include: record size, tab settings, truncation column, and uppercase or lowercase characters associated with that particular file type. The CMS Editor creates a file according to these attributes. (2) File types recognized by CMS commands; that is, commands that only search for and use particular file types or create one or more files with a particular file type.

resource. A program, a data file, a specific set of files, a device, or any other entity or a set of entities that the

user can uniquely identify for application program processing in a VM system.

resource manager. An application running in a server virtual machine that directly controls one or more VM resources. There are three categories of VM resource managers: global, local, and private.

Restructured Extended Executor (REXX) language. A general-purpose, high-level programming language, particularly suitable for EXEC procedures, XEDIT macros, or programs for personal computing. Procedures, XEDIT macros, and programs written in this language can be interpreted by the System Product Interpreter. Also, a component of VM/SP. Contrast with *CMS EXEC language* and *EXEC 2 language*.

REXX EXEC. An EXEC procedure or XEDIT macro written in the REXX language and processed by the System Product Interpreter. Synonymous with *REXX program*.

REXX program. Synonym for *REXX EXEC*.

route. A connection to another system by a logical link and one or more intermediate systems. In TSAF, a number of links and possible intermediate systems that allow the connection of one system to another.

RSCS. Remote Spooling Communications Subsystem Networking.

S

saved system. A special nonrelocatable copy of a virtual machine's virtual storage and associated registers kept on a CP-owned disk and loaded by name instead of by I/O device address. Loading a saved system by name substantially reduces the time it takes to IPL the system in a virtual machine. In addition, a saved system such as CMS can also share one or more 64K segments of reenterable code in real storage between virtual machines. This reduces the cumulative real main storage requirements and paging demands of such virtual machines.

scale. A line on the System Product Editor's (XEDIT) full-screen display, used for column reference.

screen. An illuminated display surface; for example, the display surface of a CRT. Synonymous with *physical screen*.

second-level storage. The storage that appears to be real to a virtual machine. Contrast with *first-level storage* and *third-level storage*.

segment. A contiguous 64K or 1024K area of virtual storage (not necessarily contiguous in real storage) allocated to a job or system task. VM/SP does not use 1024K segments, but supports any VM operating system that uses 1024K segments.

segment table. In System/370 virtual storage systems, a table used in DAT to control user access to virtual storage segments. Each entry indicates the length, location, and availability of a corresponding page table.

selective preventive service. The selective application of PTFs from the PUT. Contrast with *preventive service*.

server. (1) The general name for a virtual machine that provides a service for a requesting virtual machine. (2) The program that responds to a request from another computer through SRPI. Contrast with *requester*.

server-requester programming interface (SRPI). (1) A protocol between requesters and servers in an enhanced connectivity network. Includes the protocol to define a cooperative processing subsystem. (2) The interface that enables enhanced connectivity between requesters and servers in a network.

server system. A data processing system containing one or more servers providing services in response to a request from another computer.

service. Changing a product after installation. See *corrective service, local service,* and *program update service*.

session. The SNA term for a connection between two LUs. The LUs involved allocate conversations across sessions.

sever. Ending communication with another virtual machine or with the user's own virtual machine.

SFS. Shared file system.

SFS directory. A group of files. SFS directories can be arranged to form a hierarchy in which one directory can contain one or more subdirectories as well as files.

shared file system (SFS). A part of CMS that lets users organize their files into groups known as *directories* and to selectively share those files and directories with other users.

shared segment. A feature of a saved system or physical saved segment that lets one or more segments of reentrant code or data in real storage be shared among many virtual machines. For example, if a saved CMS system was generated, the CMS nucleus is shared in real storage among all CMS virtual machines loaded by name; that is, every CMS machine's segment of virtual storage maps to the same 64K of real storage. See *discontiguous saved segment* and *saved system*.

signaling attention. An indication that a user has pressed a key or keyed in a CP command to present an attention interrupt to CP or to the user's virtual machine.

simultaneous peripheral operations online (SPOOL). (1) (Noun) An area of auxiliary storage defined to temporarily hold data during its transfer between peripheral equipment and the processor. (2) (Verb) To use auxiliary storage as a buffer storage to reduce processing delays when transferring data between peripheral equipment and the processing storage of a computer.

single processor mode. In tightly coupled MP or AP systems, single processor mode lets an installation dedicate a processor to an MVS V=R virtual machine. In single processor mode, VM/SP runs in uniprocessor mode in the main processor, and the MVS V=R virtual machine runs under VM/SP in the main processor and has the exclusive use of the other processor for MP or AP operations. However, other virtual machines can operate under VM/SP concurrently with the MVS V=R virtual machine in single processor mode (not to be confused with *uniprocessor mode*).

SIO. Start I/O.

SNA. Systems Network Architecture.

source code. The input to a compiler or assembler, written in a source language. Contrast with *object code*.

source file. A file that contains source statements for such items as high-level language programs and data description specifications.

SPOOL. Simultaneous peripheral operations online.

spool file class. A one-character class associated with each virtual unit record device. For input spool files, the spool file class lets the user control which input spool files are read next; and, for output spool files, it lets the spooling operator better control or reorder the printing or punching of spool files having similar characteristics or priorities. The spool file class value can be A through Z or 0 through 9.

spooling. The processing of files created by or intended for virtual readers, punches, and printers. The spool files can be sent from one virtual device to another, from one virtual machine to another, and to real devices. See *virtual console spooling*.

spooling devices. I/O devices (card readers, punches, printers, DASD) that read input and write output.

SRPI. Server-requester programming interface.

SSP. System service program.

stack. See *console stack* and *program stack*.

stand-alone. Pertaining to an operation independent of another device, program, or system.

stand-alone dump. A dump acquired without regular system functions. For example, to obtain a CP dump when the regular system is unable to dump the machine, the stand-alone dump facility gets a CP stand-alone dump.

storage key. An indicator associated with one or more storage blocks that requires that tasks have a matching protection key in order to use the blocks.

subcommand. The commands of processors such as EDIT or System Product Editor (XEDIT) that run under CMS.

supervisor call instruction (SVC). An instruction that interrupts a program being executed and passes control to the supervisor so that it can do a specific service indicated by the instruction.

supervisor state. A state during which the processor can execute I/O and other privileged instructions. Only CP can execute in the supervisor state; all virtual machine operating systems run in problem state. Contrast with *problem state*.

SVC. Supervisor call instruction.

syntax. The rules for the construction of a command or program.

system integrity. The property of a system that is designed, implemented, and maintained to protect itself from unauthorized access.

system load. The combination of active devices, programs, and users that use the system resources of the processor and storage.

System Product Editor. The CMS facility, comprising the XEDIT command and XEDIT subcommands and macros, that lets a user create, change, and manipulate CMS files.

System Product Interpreter. The component of the VM/SP operating system that processes procedures, XEDIT macros, and programs written in the REXX language.

system programmer privilege class. The CP privilege class C user; usually, the VM/SP system programmer, who can change the contents of any real storage locations in the machine.

system restart. The restart that allows reuse of previously initialized areas. System restart usually requires less time than IPL. See *warm start*.

system service program (SSP). In ACF/TCAM, an IBM-supplied or user-supplied program that does system-oriented auxiliary functions in support of the message control program. System service programs run under control of the initiator as attached subtasks.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

T

target. One of many ways to identify a line to be searched for by the System Product Editor. A target can be specified as an absolute line number, a relative displacement from the current line, a line name, or a string expression.

task ID. A 2-byte field that uniquely defines a task within a GCS virtual machine. Task ID is sometimes combined with machine ID to uniquely identify a task within a virtual machine group.

terminal. A device, usually equipped with a keyboard and a display, capable of sending and receiving information.

third-level storage. The virtual storage created and controlled by an OS/VS or VM virtual machine. Contrast with *first-level storage* and *second-level storage*.

time-of-day (TOD) clock. A hardware feature required by VM/SP. The TOD clock is incremented once every microsecond, and provides a consistent measure of elapsed time suitable for the indication of date and time; it runs regardless of the processor state (running, wait, or stopped).

time stamp. A record containing the TOD clock value stored in its internal 32-bit binary format.

TOD clock. Time-of-day clock.

token. An eight-character symbol created by the CMS EXEC processor when it scans an EXEC procedure or EDIT macro statements. Symbols longer than eight characters are truncated to eight characters.

tokenized PLIST (parameter list). A string of doubleword aligned parameters occupying successive doublewords.

trace table. Synonym for *CP trace table*.

Transparent Services Access Facility (TSAF). A component of VM/SP that handles communication between systems by letting APPC/VM paths span multiple VM systems. TSAF lets a source program connect to a target program by specifying a name that the target has made known, instead of specifying a user ID and node ID.

TSAF. Transparent Services Access Facility.

TSAF collection. A group of VM processors, each with a TSAF virtual machine, connected by CTC, binary synchronous lines, or LANs.

TSAF virtual machine. The virtual machine that lets user programs connect to and communicate with virtual machines on different VM systems.

U

UCR. User class restructure.

uniprocessor mode. This term indicates that there is only one processor in the physical configuration, or that VM/SP uses the facilities of one processor in an AP or MP system (not to be confused with *single processor mode*).

universal class card reader. A virtual card reader that can read any class of reader, printer, or punch files spooled or transferred to it.

user. Anyone who requests the services of a computing system.

user class. A privilege category assigned to a virtual machine user in the user's directory entry; each class specified allows access to a logical subset of all the CP commands. See *privilege class*.

user class restructure (UCR). The extension of the class structure of CP instructions from 8 to 32 classes for each user, command, and diagnose code within the system. This extension allows the installation greater flexibility in authorizing CP instructions.

user data. In reference to a file pool, any data that resides in storage groups 2 through 32767.

user ID. User identification.

user input area. On a display device, the lines of the screen where the user is required to key in command or data lines. See *display mode*, *input line*, and *line mode*.

user modification. Any change that a user originates for a product or component.

user program. A transaction program that requests a service from a resource manager program. User programs reside in requester virtual machines.

user-written CMS command. Any CMS file created by a user that has a file type of MODULE or EXEC. Such

a file can be executed as if it were a CMS command by issuing its file name, followed by any operands or options expected by the program or EXEC procedure.

V

virtual address. The address of a location in virtual storage. A virtual address must be translated into a real address in order to process the data in processor storage.

virtual card reader. CP's simulation on disk of a real card reader. A virtual card reader can read card, punch, or print records of up to 151 characters in length. The virtual device type and I/O device address are usually defined in the VM/SP directory. See *spool file class* and *universal class card reader*.

virtual console. • A console simulated by CP on a terminal such as a 3270. The virtual device type and I/O address are defined in the VM/SP directory entry for that virtual machine.

virtual console spooling. The writing of console I/O on disk as a printer spool file instead of, or in addition to, having it typed or displayed at the virtual machine console. The console data includes messages, responses, commands, and data from or to CP and the virtual machine operating system. The user can invoke or terminate console spooling at anytime. When the console spool file is closed, it becomes a printer spool file. Synonymous with *console spooling*.

virtual disk. A logical subdivision (or all) of a physical disk storage device that has its own address, consecutive storage space for data, and an index or description of the stored data so that the data can be accessed. A virtual disk is also called a minidisk. See *disk*.

virtual interval timer assist. A hardware assist function, available only on a processor, that has ECPS. It provides, if desired, a hardware updating of each virtual machine's interval timer at location X'50'.

virtual machine (VM). A functional equivalent of a real machine.

virtual machine assist (VMA). A hardware feature available on certain VM/SP-supported System/370 models, that causes a significant reduction in the real supervisor state time that VM/SP uses to control the operation of virtual storage systems such as VSE, DOS/VS and OS/VS and, to a lesser extent, CMS, DOS, and OS when running under VM/SP. VM/SP supervisor state time is reduced because the VMA feature, instead of VM/SP, intercepts and handles interruptions caused by SVCs, other than SVC 76, and certain privileged instructions. See *CP assist, expanded virtual machine assist, Extended Control Program Support (ECPS:VM/370)*, and *virtual interval timer assist*.

virtual machine communication facility (VMCF). A CP function that provides a method of communication and data transfer between virtual machines operating under the same VM/SP system.

virtual machine control block (VMBLOK). The primary control block for many activities related to a single virtual machine. This block contains, for each virtual machine, the following types of information: the dispatch and priority level of the virtual machine, the virtual machine's processor registers, preferred virtual machine options currently in effect, and information concerning all other significant activities.

virtual machine group. The concept in GCS of two or more virtual machines associated with each other through the same named system (for example, IPL GCS1). Virtual machines in a group share common read/write storage and can communicate with one another through facilities provided by GCS. Synonymous with *group*.

Virtual Machine/System Product (VM/SP). An IBM licensed program that manages the resources of a single computer so that multiple computing systems appear to exist. Each virtual machine is the functional equivalent of a *real* machine.

virtual printer (or punch). A printer (or card punch) simulated on disk by CP for a virtual machine. The virtual device type and I/O address are usually defined in the VM/SP directory entry for that virtual machine.

virtual=real option. A VM/SP performance option that lets a virtual machine run in VM/SP's virtual=real area. This option eliminates CP paging and, optionally, CCW translation for this virtual machine. Synonymous with *V=R*.

virtual storage. Storage space that can be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computing system and by the amount of auxiliary storage available, and not by the actual number of main storage locations.

virtual storage access method (VSAM). An access method for direct or sequential processing of fixed and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number.

virtual storage extended (VSE). The generalized term that indicates the combination of the DOS/VSE system control program and the VSE/Advanced Functions program product. Note that in certain cases, the term

DOS is still used as a generic term; for example, disk packs initialized for use with VSE or any predecessor DOS or DOS/VS system are sometimes called DOS disks. Also note that the DOS-like simulation environment provided under the VM/SP CMS component and CMS/DOS exists on VM/SP and VM/SP HPO program products and continues to be called CMS/DOS.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in a computer network. It provides single-domain, multiple-domain, and multiple-network capability. VTAM runs under MVS, OS/VS1, VM/SP, and VSE.

VM. Virtual machine.

VMA. Virtual machine assist.

VMBLOK. Virtual machine control block.

VMCF. Virtual machine communication facility.

VM/SP. Virtual Machine/System Product.

VM/SP directory. A CP disk file that defines each virtual machine's typical configuration; the user ID, password, regular and maximum allowable virtual storage, CP command privilege class or classes allowed, dispatching priority, logical editing symbols to be used, account number, and CP options desired. Synonymous with *CP directory*.

volid. Volume identifier.

volume identifier (volid). The volume identification label for a disk.

V=R. Synonym for *virtual=real option*.

VSAM. Virtual storage access method.

VSCS. VTAM SNA Console Support.

VSE. Virtual storage extended.

VTAM. Virtual Telecommunications Access Method.

W

warm start. (1) The result of an IPL that does not erase previous system data. (2) The automatic reinitialization of the VM/SP control program that occurs if the control program cannot continue processing. Closed spool files and the VM/SP accounting information are not lost. Contrast with *checkpoint (CKPT) start, cold start*, and *force start*.

wide area network (WAN). A network that provides communication services to a geographic area larger than that served by an LAN. Contrast with *local area network (LAN)*.

wrap spool file. A wrap spool file is established when the CPTRAP invoker issues CPTRAP START with the WRAP option. The size of the wrap spool file is determined by the file size information provided with the CPTRAP START WRAP nnnnn command. (*nnnnn* is the number of 4K blocks of records.) Records will be added to the spool file until the specified SPOOL size limit is reached. Then, newer records replace older records in the spool file thereby using the same spool area over again.

X

XEDIT. See *System Product Editor*.

Z

zap. To modify or dump an individual text file, using the ZAP command or the ZAPTEXT EXEC.

Numerics

3

3262. Refers to the IBM 3262 Printer, Models 1 and 11.

3270. Refers to a series of IBM display devices; for example, the IBM 3275, 3276 Controller Display Station, 3277, 3278, and 3279 Display Stations, the 3290 Information Panel, and the 3287 and 3286 printers. A specific device type is used only when a distinction is required between device types. Information about display terminal usage also refers to the IBM 3138, 3148, and 3158 Display Consoles when used in display mode, unless otherwise noted.

3289. Refers to the IBM 3289 Model 4 Printer.

3330. Refers to the IBM 3330 Disk Storage Device.

3340. Refers to the IBM 3340 Direct Access Storage Device.

3350. Refers to the IBM 3350 Direct Access Storage Device when used in native mode.

3375. Refers to the IBM 3375 Direct Access Storage Device.

3380. Refers to the IBM 3380 Direct Access Storage Device.

3422. Refers to the IBM 3422 Magnetic Tape Subsystem.

370x. Refers to the IBM 3704/3705 Communication Controllers.

3725. Refers to the IBM 3725 Communication Controllers.

3800. Refers to the IBM 3800 Printing Subsystems. A specific device type is used only when a distinction is required between device types.

3880. Refers to the IBM 3880 Storage Control Units.

4

4245. Refers to the IBM 4245 Printer.

4248. Refers to the IBM 4248 Printer.

9

9332. Refers to the IBM 9332 Direct Access Storage Device, Model 400.

9335. Refers to the IBM 9335 Direct Access Storage Device, Models A01 and B01.

9347. Refers to the IBM 9347 Tape Drive.

9370. Refers to a series of processors, namely the IBM 9373 Models 20 and 30, the IBM 9375 Models 40, 50, and 60, and the IBM 9377 Models 80 and 90.

Bibliography

Here is a list of IBM books that can help you use your system. If you don't see the book you want in this list, you might want to check the *IBM System/370, 30xx, 4300, and 9370 Processors Bibliography*, GC20-0001.

Prerequisite Publications

IBM System/360 Principles of Operation, GA22-6821

IBM System/370 Principles of Operation, GA22-7000

Virtual Machine/System Product (VM/SP):

Application Development Reference for CMS, SC24-5284
Application Development Guide for CMS, SC24-5286
CMS Command Reference, SC19-6209
CMS User's Guide, SC19-6210
CP General User Command Reference, SC19-6211
CP System Command Reference, SC24-5402
Release 6 Guide, SC24-5368
System Product Editor Command and Macro Reference, SC24-5221
System Product Editor User's Guide, SC24-5220
System Product Interpreter Reference, SC24-5239
System Product Interpreter User's Guide, SC24-5238

Corequisite Publications

Virtual Machine/System Product (VM/SP):

Administration, SC24-5285
Application Development Guide for CMS, SC24-5286
CMS Data Areas and Control Blocks, LY24-5221
CMS Diagnosis Reference, LY20-0893
CP Data Areas and Control Blocks, LY24-5220
CMS Shared File System Administration, SC24-5367
Connectivity Programming Guide and Reference, SC24-5377
Connectivity Planning, Administration, and Operation, SC24-5378
CP Diagnosis Reference, LY20-0892
Group Control System Command and Macro Reference, SC24-5250
Installation Guide, SC24-5237
Interactive Problem Control System Guide and Reference, SC24-5260
Library Guide and Master Index, SC19-6207
Operator's Guide, SC19-6202
Planning Guide and Reference, SC19-6201
System Messages and Codes, SC19-6204
System Messages Cross-Reference, SC24-5264
Service Routines Program Logic, LY20-0890
Problem Determination Summary, SX24-5224

Virtual Machine (VM):

CP Trace Table (Poster), SX24-5225
Running Guest Operating Systems, GC19-6212
System Facilities for Programming, SC24-5288

VM/SP Remote Spooling Communications Subsystem (RSCS) Networking Version 2:

Diagnosis Reference, LY24-5228
General Information, SH24-5055

Operation and Use, SH24-5058
Planning and Installation, SH24-5057
Program Reference and Operations Manual, SH24-5005

Virtual Machine/System Product (VM/SP) Pass-Through Facility:

Managing and Using, SC24-5374

Customer Information Control System for VM (CICS/VM): Problem Determination, LY33-0533

IBM Field Engineering Programming System: General Information, G229-2228

IBM System/370 and 4300 Processor Bibliography, GC20-0001

IBM Vocabulary for Data Processing, Telecommunications, and Office Systems, GC20-1699

VSE/VSAM Programmer's Reference, SC24-5145

Notes:

1. References in text to titles of publications are given in abbreviated form.
2. The *VM/SP Library Guide and Master Index*, GC19-6207, describes all the VM/SP books and contains a master index to all the books in the VM/SP library.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Numerics

VM/SP
Diagnosis Guide
Order No. LY24-5241-01

READER'S COMMENT FORM

Is there anything you especially like or dislike about this book? Feel free to comment on specific errors or omissions, accuracy, organization, or completeness of this book.

If you use this form to comment on the online HELP facility, please copy the top line of the HELP screen.

_____ _____ _____ Help Information line ____ of ____

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you, and all such information will be considered nonconfidential.

Note: Do not use this form to report system problems or to request copies of publications. Instead, contact your IBM representative or the IBM branch office serving you.

Would you like a reply? ___YES ___NO

Please print your name, company name, and address:

_____

_____

_____

_____

IBM Branch Office serving you: _____

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us.

LY24-5241-01

Reader's Comment Form

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, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
DEPARTMENT G60
PO BOX 6
ENDICOTT NY 13760-9987

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

Program Number
5664-167

File Number
S370/4300-37

Printed in U.S.A.