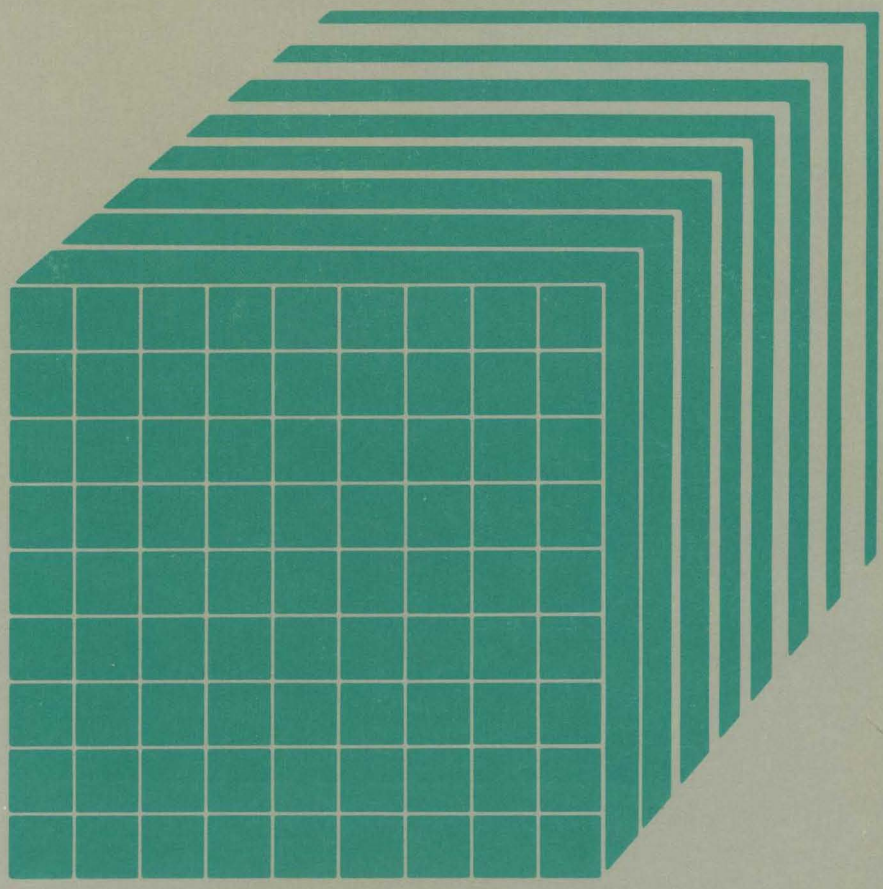
IBM

# Virtual Machine/ System Product

# System Programmer's Guide

Release 3

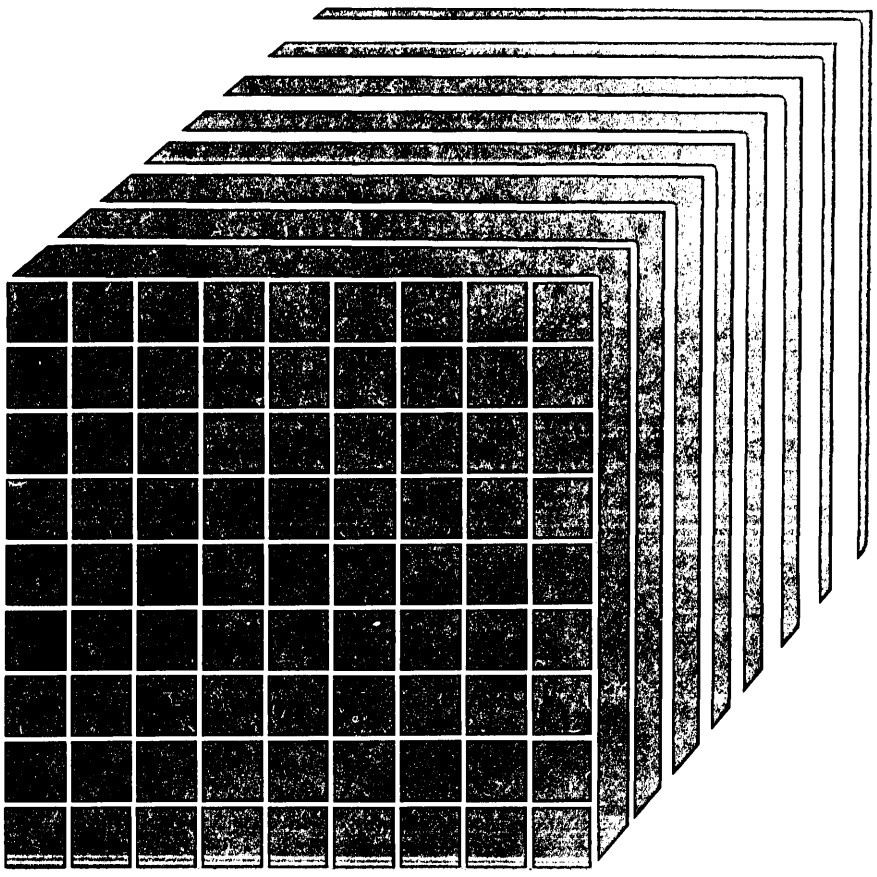


IBM

# Virtual Machine/ System Product

# System Programmer's Guide

**Release 3**



**Third Edition (August 1983)**

This edition, SC19-6203-2, applies to Release 3 of IBM Virtual Machine/System Product (VM/SP) unless otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information contained herein; before using this publication in connection with the operation of IBM systems, consult the *IBM System/370 and 4300 Processors Bibliography,* GC20-0001, for the editions that are applicable and current.

Technical changes and additions to the text and illustrations are indicated by a vertical bar to the left of the change.

**Summary of Changes**

For a detailed list of changes, see page iii.

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 program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

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

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 Programming Publications, Dept. G60, P.O. Box 6, Endicott, New York, U.S.A. 13760. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

© Copyright International Business Machines Corporation 1980, 1981, 1982, 1983

# Summary of Changes

**Summary of Changes**
**for SC19-6203-2**
**for VM/SP Release 3**

*Programmable Operator Facility*

Several enhancements to the programmable operator facility added are:

- Message routing with nicknames
- Remote node availability
- Enhanced text comparison
- EXEC action routines
- LOG recording and error handling

*PER*

Problem determination capability is greatly extended and enhanced by the new CP command, PER.

*DASD Block I/O System Service*

The DASD block I/O system service allows a virtual machine fast, device-independent asynchronous access to fixed size blocks on CMS formatted virtual DASD I/O devices.

*IUCV*

Inter-User Communication Vehicle (IUCV) extensions provide:

- SEND and REPLY extensions
- An extended mask capability for control interrupts
- An expanded trace capability to record all IUCV operations
- A macro option to initialize the parameter list
- Support for the DASD block I/O system service.

*The IBM 3088 Multisystem Communications Unit*

The IBM 3088 Multisystem Communications Unit interconnects multiple systems using block multiplexer channels. The 3088 uses an unshared subchannel for each unique address and is fully compatible with existing channel-to-channel adapter protocol.

*CMS IUCV support*

Support for IUCV communication has been introduced into CMS. This support allows multiple programs within a virtual machine to use IUCV functions. Included is the ability to initialize a CMS machine for IUCV communication and to invoke IUCV functions via new CMS macros. These macros also allow the user to specify path-specific exits for IUCV external interrupts.

*CMS ABEND exits*

A general CMS abnormal exit capability is provided so that user programs may specify the address of a routine to get control before CMS ABEND recovery begins. An exit is established and cleared through a new CMS macro.

*Enhanced immediate command support*

The immediate command capability of CMS is extended by allowing users to define their own immediate commands.

*Enhanced VSAM support*

CMS supports VSE/VSAM Release 3 which includes significant enhancements designed to improve catalog reliability and integrity while providing additional serviceability and usability. VSE/VSAM Release 2 is not supported.

*Miscellaneous*

Changes to the DIAGNOSE zero interface provide the time zone differential from Greenwich Mean Time.

DIAGNOSE X'8C' allows a virtual machine to access device dependent information without having to issue a WRITE STRUCTURE FIELD QUERY REPLY.

CMSSEG has been eliminated and the code was merged into the CMS Nucleus.

The Remote Spooling Communications Subsystem (RSCS) section of this manual has been removed as it pertained to RSCS as a component of VM/370. Now, any reference to RSCS in this manual applies to the RSCS Networking Programming Product, and information can be found in the *VM/SP Remote Spooling Communications Subsystem Networking Program Reference and Operations Manual*, SH24-5005.

A newly added appendix lists and describes the CMS macros applicable to VM/SP.

Minor technical and editorial changes have been made throughout this publication.

**Summary of Changes**
**for SC19-6203-1**
**as Updated by SN24-5736**

*Missing Interrupt Handler*

The missing interrupt detector has been extended so that CP not only detects missing interrupt conditions, but also attempts to correct them. CP informs the system operator whether or not the corrective action was successful.

To help give you optimum system availability, the missing interrupt handler allows you to vary the time interval allowed for I/O completion for the supported devices.

*3880 Speed Matching Buffer (Feature #6560)*

The 3880 Speed Matching Buffer Feature for the IBM 3375 uses a 16K-byte storage buffer to modify the direct access data transfer path between the 3375 and the multiplexer channel. The feature allows attachment of the 3375 Direct Access Storage Device, with its 1.859 megabytes per second data rate, to block multiplexer channels with data rates as low as 1.5 megabytes per second, as well as to high speed multiplexer channels.

*Miscellaneous*

The Programmable Operator Facility section of this publication has been rewritten to include minor technical and editorial changes.

**Summary of Changes**
**for SC19-6203-1**
**for VM/SP Release 2**

*Programmable Operator Facility*

This facility provides the capability to: log messages, suppress messages, redirect messages, execute messages, or preprogram message responses. The capabilities are under control of an editable message routine table in a CMS file.

Inter-User Communication Vehicle (IUCV) enhancements for message handling are also included.

*CMS Nucleus Restructure, and Removal of the CMS Tokenization Eight-Byte Restriction*

The restructured nucleus provides a CMS system that is more flexible and extendable for development, serviceability, and maintenance purposes.

The eight-byte tokenizer restriction has been removed for parameter passing.

*Trace Table Recording Facility*

This facility allows service personnel and system programmers to create a chronological READER spool file of CP trace entries by type, VMBLOK address, interrupt code, and device address.

*Miscellaneous*

Minor technical and editorial changes have been made throughout this publication.

# Preface

This publication describes how to debug VM/SP and how to modify, extend, or implement Control Program (CP) and Conversational Monitor System (CMS) functions. This information is intended for system programmers, system analysts, and programming personnel.

This publication consists of three parts and three appendixes.

"Part 1. Control Program (CP)" contains an introductory and functional description of CP as well as guidance in implementing some CP features.

"Part 2. Conversational Monitor System (CMS)" contains an introductory and functional description of CMS including how CMS handles interrupts and SVCs, structures its nucleus and its storage, and manages free storage. Information on saving the CMS system and implementing the Batch Facility is also included.

"Part 3. Debugging with VM/SP" discusses the CP and CMS debugging tools and procedures to follow when debugging. This part is logically divided into three sections. The first section, "Introduction to Debugging", tells you how to identify a problem and lists guidelines to follow to find the cause. The second section describes the CP debugging commands and utilities, debugging CP in a virtual machine, the internal trace table, and restrictions. A detailed description of CP dump reading is also included. The third section, "Debugging with CMS", describes the CMS debugging commands and utilities, load maps, and restrictions and tells you what fields to examine when reading a CMS dump.

"Appendix A: System/370 Information" describes the System/370 extended PSW and extended control register usage.

"Appendix B: VM Monitor Tape Format and Contents" describes the format and contents of data records for classes and codes of MONITOR CALL.

"Appendix C: CMS Macro Library" lists and describes the CMS macros applicable to VM/SP.

## *Terminology*

Some of the following convenience terms are used throughout this publication:

- Throughout this publication, the term "VM/SP" refers to the VM/SP program package when you use it in conjunction with VM/370 Release 6. The terms "CP" and "CMS" refer to the VM/370 components enhanced by the functions included in the VM/SP package. Any reference to "RSCS," unless otherwise noted, is to the RSCS Networking Program Product (5748-XP1). Any reference to "IPCS," unless otherwise noted, is to the IPCS Extended Program Product (5748-SA1).

  When you install and use VM/SP in conjunction with the VM/SP Release 6 System Control Program (SCP), it becomes a functional operating system that provides extended features to the Control Program (CP) and Conversational Monitor System (CMS) components of VM/370 Release 6. VM/SP adds *no* additional functions to the Remote Spooling Communications Subsystem (RSCS) and the Interactive Problem Control System (IPCS) components of VM/370. However, you can appreciably expand the capabilities of these

components in a VM/SP system by installing the RSCS Networking program product (5748-XP1) and the VM/IPCS Extension program product (5748-SA1).

- Unless otherwise noted, the term VSE refers to the combination of the DOS/VSE system control program and the VSE/Advanced Functions program product. 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/VSE system may be referred to as DOS disks.

  The DOS-like simulation environment provided under the CMS component of the VM/System Product, continues to be referred to as CMS/DOS.

- Unless otherwise noted, the term "EXEC" refers to EXECs using the System Product Interpreter, EXEC 2, or CMS EXEC languages.

- Unless otherwise noted, the term "System/370" applies to the 4300 and 303X processors.

  The following terms in this publication refer to the indicated support devices:

- "2305" refers to IBM 2305 Fixed Head Storage, Models 1 and 2.

- "3262" refers to the IBM 3262 Printer, Models 1, 5, and 11.

- "3270" refers to a series of display devices, namely, the IBM 3275, 3276 (referred to as a Controller Display Station), 3277, 3278, and 3279 Display Stations. A specific device type is used only when a distinction is required between device types.

  Information about display terminal usage also applies to the IBM 3138, 3148, and 3158 Display Consoles when used in display mode, unless otherwise noted.

  Any information pertaining to the IBM 3284 or 3286 Printer also pertains to the IBM 3287, 3288, and 3289 printers, unless otherwise noted.

- "3330" refers to the IBM 3330 Disk Storage, Models 1, 2, or 11; the IBM 3333 Disk Storage and Control, Models 1 or 11; and the 3350 Direct Access Storage operating in 3330 compatibility mode.

- "3340" refers to the IBM 3340 Direct Access Storage Facility and the 3344 Direct Access Storage.

- "3350" refers to the IBM 3350 Direct Access Storage Device when used in native mode.

- "3375" refers to the IBM 3375 Direct Access Device.

- "3380" refers to the IBM 3380 Direct Access Storage. The Speed Matching Buffer Feature (No. 6550) for the 3380 supports the use of extended count-key-data channel programs.

- "3430" refers to the IBM 3430 Magnetic Tape Subsystem.

- "370X" refers to IBM 3704 and 3705 Communications Controllers.

- "3705" refers to the 3705 I and the 3705 II unless otherwise noted.
- "2741" refers to the IBM 2741 and the IBM 3767, unless otherwise specified.
- "3066" refers to the IBM 3066 System Console.
- "3800" refers to the IBM 3800 Printing Subsystem.
- "3081" refers to the IBM 3081 Processor Unit model D16.
- "3088" refers to the IBM 3088 Multisystem Communications Unit (MCU) Models 1 and 2.
- "4245" refers to the IBM 4245 Line Printer.
- "4250" refers to the IBM 4250 Printer.

An expanded glossary is available in the *Virtual Machine/System Product: Library Guide and Master Index*, GC19-6207.

Knowledge of Assembler Language and experience with programming concepts and techniques are prerequisite to using this publication.

References to a program that produces a standalone dump occur in several places in this publication. One such program is the BPS Storage Print program, Program No. 360P-UT-056.

## *Prerequisite Publications*

*IBM System/360 Principles of Operation*, GA22-6821

*IBM System/370 Principles of Operation*, GA22-7000

*Virtual Machine/System Product: Operating Systems in a Virtual Machine*, GC19-6212

## *Corequisite Publications*

Knowledge of the commands and system functions of CP, CMS, and RSCS is corequisite.

*Virtual Machine/System Product:*

*Planning Guide and Reference*, SC19-6201

*Installation Guide*, SC24-5237

*CP Command Reference for General Users*, SC19-6211

*CMS Command and Macro Reference*, SC19-6209

*CMS User's Guide*, SC19-6210

*Operator's Guide*, SC19-6202

*Terminal Reference*, SC19-6206

## *Supplemental Publications*

*OS/VS Data Management Macro Instructions*, GC26-3793

*OS/VS Supervisor Service and Macro Instructions*, GC27-6979

*IBM 2821 Control Unit Component Description*, GA24-3312

*IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide*, GA24-3543

*IBM 3262 Printers 1 and 11 Component Description*, GA24-3733

*IBM 3270 Information Display System Library User's Guide*, GA23-0058

*OS/VS Linkage Editor and Loader*, GC26-3813

*Introduction to the IBM 3704 and 3705 Communications Controllers*, GA27-3051

*IBM 3704 and 3705 Communications Controllers Operator's Guide*, GA27-3055

*IBM Virtual Machine Facility/370: Performance/Monitor Analysis Program*, SB21-2101

*OLTSEP and Error Recording Guide*, SC19-6205

This publication contains a description of CPEREP. CPEREP is a CMS command that invokes OS/VS EREP operands to produce statistical reports from error recording data of hardware and software errors.

*Environmental Recording Editing and Printing (EREP) Program*, GC28-1178

This publication contains a detailed description of the CPEREP operands, and is required in order to make use of CPEREP.

*VM/SP Remote Spooling Communications Subsystem Networking Program Reference and Operations Manual*, SH24-5005

*VM/SP Data Areas and Control Block Logic*,

*Volume 1 Control Program (CP)*, LY24-5220

*Volume 2 Conversational Monitor System (CMS)*, LY24-5221

*VM/SP System Logic and Problem Determination*,

*Volume 1 Control Program (CP)*, LY20-0892

*Volume 2 Conversational Monitor System (CMS)*, LY20-0893

*VM/SP Remote Spooling Communication Subsystem Networking Logic*, LY24-5208

If the IBM 3767 Communication Terminal is used by the system programmer as a virtual machine console, the *IBM 3767 Operator's Guide*, GA18-2000 is also a corequisite publication.

If the IBM 3850 Mass Storage System is attached to the VM/SP system, the following are corequisite publications:

*IBM 3850 Mass Storage System (MSS) Introduction and Preinstallation Planning*, GA32-0038

*OS/VS Message Library: Mass Storage System (MSS) Messages*, GC38-1000

*IBM 3850 Mass Storage System (MSS) Principles of Operation: Theory*, GA32-0035

*IBM 3850 Mass Storage System (MSS) Principles of Operation: Reference*, GA32-0036

*OS/VS Mass Storage System (MSS) Services: General Information*, GC35-0016

*OS/VS Mass Storage System (MSS) Services: Reference Information*, GC35-0017

*Operator's Library: IBM 3850 Mass Storage System (MSS) Under OS/VS*, GC35-0014.

**Note:** References in text to titles of corequisite VM/SP publications are given in abbreviated form.

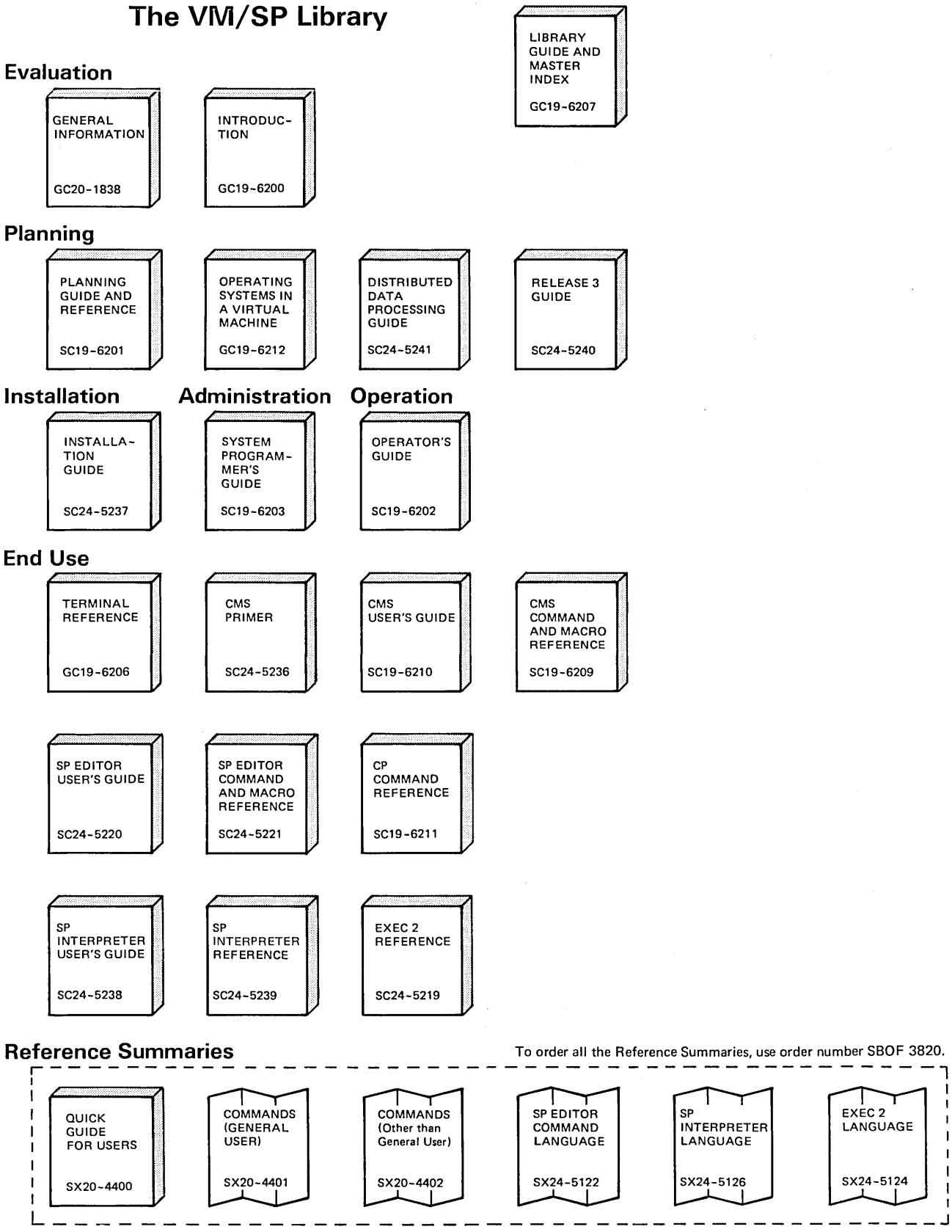



**Figure 1 (Part 1 of 2). The VM/SP Library**

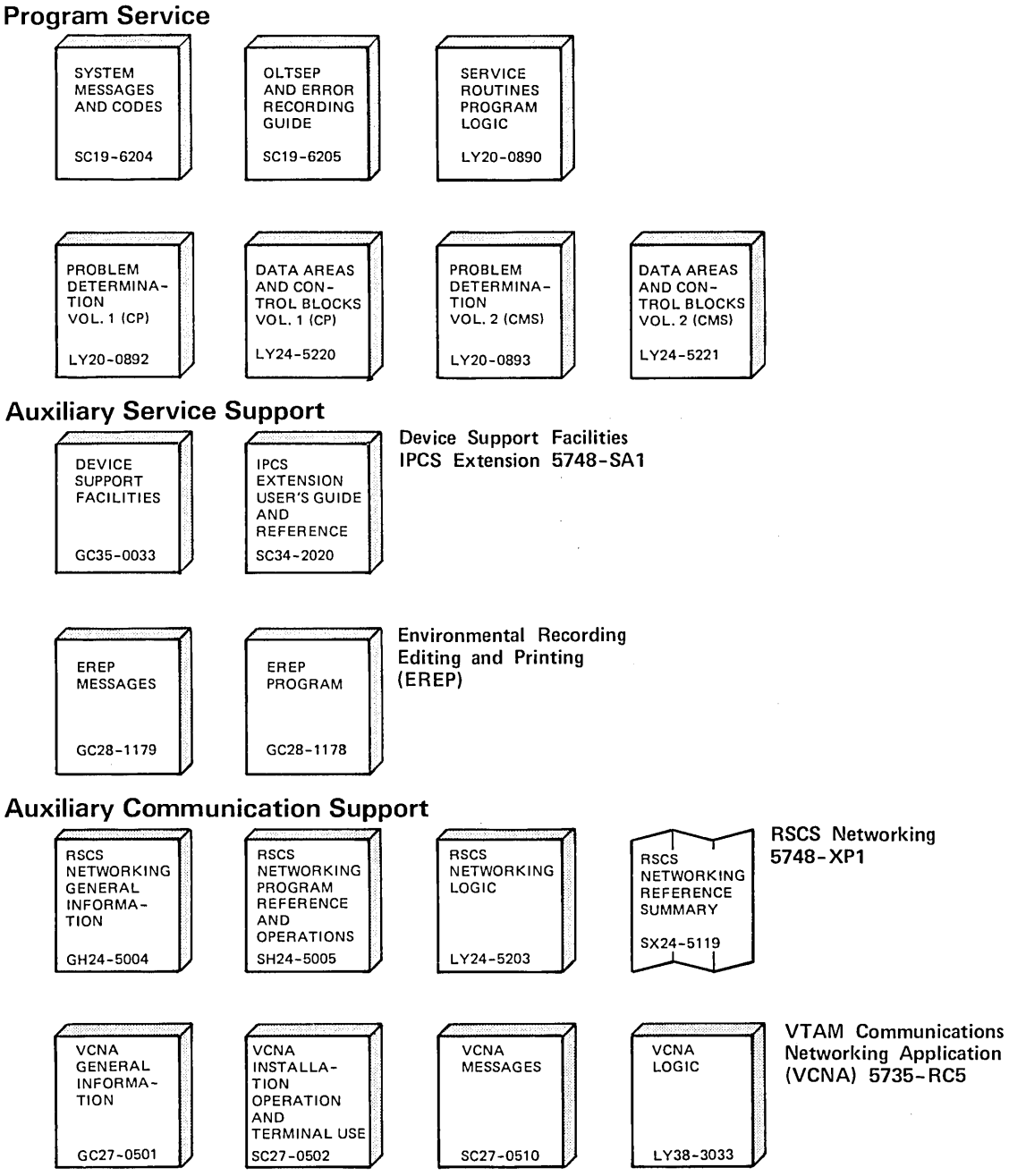



**Figure 1 (Part 2 of 2). The VM/SP Library**

# Contents

# Figures

# Part 1. Control Program (CP)

Part 1 contains the following information:

- Introduction to VM/SP
- Program States
- Using Processor Resources
- Interruption Handling
- Functional Information
- Performance Guidelines
- Virtual Machine Assist Feature
- VM/370 Extended Control-Program Support
- VM/VS Handshaking
- Performance Observation and Analysis
- Accounting Information
- Generating Named Systems and Saving Systems
- The Virtual Machine Communication Facility
- The Inter-User Communications Vehicle
- SNA Virtual Console Support
- The Message System Service
- The DASD Block I/O System Service
- The Special Message Facility
- The Single Image Consol Facility
- VM/SP Use of the IBM 3850 MSS
- The Logical Device Support Facility
- Timers
- CP in Attached Processor and Multiprocessor Modes
- DIAGNOSE Instruction
- CP Conventions
- How To Add a Console Function
- How To Add a New Print or Forms Buffer Image
- The IBM 3800 Printing Subsystem
- Journaling Logon, Autolog, and Link Commands
- Suppressing Passwords

The VM/SP Control Program manages the resources of a single computer in such a manner that multiple computing systems appear to exist. Each "virtual" computing system, or virtual machine, is the functional equivalent of an IBM System/370.

A virtual machine is configured by recording appropriate information in the VM/SP directory. The virtual machine configuration includes counterparts of the components of a real IBM System/370:

- A virtual operator's console
- Virtual storage
- A virtual processor
- Virtual I/O devices

CP makes these components appear real to whichever operating system is controlling the work flow of the virtual machine.

The virtual machines operate concurrently via multiprogramming techniques. CP overlaps the idle time of one virtual machine with execution in another.

Each virtual machine is managed at two levels. The work to be done by the virtual machine is scheduled and controlled by some System/360 or System/370 operating system. The concurrent execution of multiple virtual machines is managed by the Control Program.

VM/SP performs some functions differently when running in attached processor or multiprocessor mode. For a description of the additional processing performed, see the *VM/SP System Logic and Problem Determination Guide.*

## Introduction to the VM/SP Control Program

A virtual machine is created for a user when he logs on VM/SP, on the basis of information stored in his VM/SP directory entry. The entry for each user identification includes a list of the virtual input/output devices associated with the particular virtual machine.

Additional information about the virtual machine is kept in the VM/SP directory entry. Included are the VM/SP command privilege class, accounting data, normal and maximum virtual storage sizes, dispatching priority, and optional virtual machine characteristics such as extended control mode.

The Control Program supervises the execution of virtual machines by permitting only problem state execution except in its own routines, and receiving control after all real computing system interrupts. CP intercepts each privileged instruction and simulates it if the current program status word of the issuing virtual machine indicates a virtual supervisor state; if the virtual machine is executing in virtual problem state, the attempt to execute the privileged instruction is reflected to the virtual machine as a program interrupt. All virtual machine interrupts (including those caused by attempting privileged instructions) are first handled by CP, and are reflected to the virtual machine if a similar interrupt would have occurred on a real machine.

### *Virtual Machine Time Management*

The real processor simulates multiple virtual processors. Virtual machines that are executing in a conversational manner are given access to the real processor more frequently than those that are not; these conversational machines are assigned the smaller of two possible time slices. CP determines execution characteristics of a virtual machine at the end of each time slice on the basis of the recent frequency of its console requests or terminal interrupts. The virtual machine is queued for subsequent processor use according to whether it is a conversational or nonconversational user of system resources.

A virtual machine can gain control of the processor only if it is not waiting for some activity or resource. The virtual machine itself may enter a virtual wait state after an input/output operation has begun. The virtual machine cannot gain control of the real processor if it is waiting for a page of storage, if it is waiting for an input/output operation to be translated and started, or if it is waiting for a CP command to finish execution.

A virtual machine can be assigned a priority of execution. Priority is a parameter affecting the execution of a particular virtual machine as compared with other virtual machines that have the same general execution characteristics. Priority is a parameter in the virtual machine's VM/SP directory entry. The system operator can reset the value with the privilege class A SET PRIORITY command.

### *Virtual Machine Storage Management*

The normal and maximum storage sizes of a virtual machine are defined as part of the virtual machine configuration in the VM/SP directory. You may redefine virtual storage size to any value that is a multiple of 4K and not greater than the maximum directory-defined value. VM/SP implements this storage as virtual storage. The storage may appear as paged or unpaged to the virtual machine, depending upon whether or not the extended control mode option was specified for that virtual machine. This option is required if operating systems that control virtual storage, such as OS/VS1, VM/370 or VM/SP are run in the virtual machine.

Storage in the virtual machine is logically divided into 4096-byte areas called pages. A complete set of segment and page tables is used to describe the storage of each virtual machine. These tables are updated by CP and reflect the allocation of virtual storage pages to blocks of real storage. These page and segment tables allow virtual storage addressing in a System/370 machine. Storage in the real machine is logically and physically divided into 4096-byte areas called page frames.

Only referenced virtual storage pages are kept in real storage, thus optimizing real storage use. Further, a page can be brought into any available page frame; the necessary relocation is done during program execution by a combination of VM/SP and dynamic address translation on the System/370. The active pages from all logged on virtual machines and from the pageable routines of CP compete for available page frames. When the number of page frames available for allocation falls below a threshold value, CP determines which virtual storage pages currently allocated to real storage are relatively inactive and initiates suitable page-out operations for them.

CP keeps track of where each virtual machine's page zero resides. The normal way CP does this is to issue a TRANS macro, that checks for page residency (LRA) and demands a page-in if the page is not in real storage. CP checks an in-storage

pointer in the VMBLOK; the pointer contains the address of the virtual machine's page zero if the page is resident. If the page is resident, CP bypasses issuing the TRANS macro, thus eliminating unnecessary LCTL and LRA instructions.

Inactive pages are kept on a direct access storage device. If an inactive page has been changed at some time during virtual machine execution, CP assigns it to a paging device, selecting the fastest such device with available space. If the page has not changed, it remains allocated in its original direct access location and is paged into real storage from there the next time the virtual machine references that page. A virtual machine program can use the DIAGNOSE instruction to tell CP that the information from specific pages of virtual storage is no longer needed; CP then releases the areas of the paging devices which were assigned to hold the specified pages.

Paging is done on demand by CP. This means that a page of virtual storage is not read (paged) from the paging device to a real storage block until it is actually needed for virtual machine execution. CP makes no attempt to anticipate what pages might be required by a virtual machine. While a paging operation is performed for one virtual machine, another virtual machine can be executing. Any paging operation initiated by CP is transparent to the virtual machine.

If the virtual machine is executing in extended control mode with translate on, then two additional sets of segment and page tables are kept. The virtual machine operating system is responsible for mapping the virtual storage created by it to the storage of the virtual machine. CP uses this set of tables and the page and segment tables created for the virtual machine at logon time to build shadow page tables for the virtual machine. These shadow tables map the virtual storage created by the virtual machine operating system to the storage of the real computing system. The tables created by the virtual machine operating system may describe any page and segment size permissible in the IBM System/370.

### Storage Protection

Storage keys protect information in real storage from unauthorized use. A storage key contains a four bit control field that is associated with an area of real storage. When VM/SP is executing natively, each *2K* area of storage is protected by one storage key.

VM/SP contains support that allows it to execute as a guest virtual machine on a processor that uses single key real storage frames. Single key storage frames associate one storage key for each *4K* area of storage. VM/SP does not run natively on processors that have single key storage frames; however, under control of the VM/SP High Performance Option program product, VM/SP executes as a guest virtual machine operating system.

When VM/SP High Performance Option (Release 2 or subsequent release) is controlling the processor equipped with single key storage frames, the program product simulates for the guest, virtual storage that resembles the type of real storage installed on the processor. If the storage simulated for the VM/SP guest requires 4K storage protection keys, VM/SP issues two key instructions to the referenced storage frame.

VM/SP provides both fetch and store protection for real storage. The contents of real storage are protected from destruction or misuse caused by erroneous or unau-

thorized storing or fetching by the program. Storage is protected from improper storing or from both improper storing and fetching, but not from improper fetching alone.

When protection applies to a storage access, the key in storage is compared with the protection key associated with the request for storage access. A store or fetch is permitted only when the key in storage matches the protection key.

When a store access is prohibited because of protection, the contents of the protected location remain unchanged. On fetching, the protected information is not loaded into an addressable register, moved to another storage location, or provided to an I/O device.

When a processor access is prohibited because of protection, the operation is suppressed or terminated, and a program interruption for a protection exception takes place. When a channel access is prohibited, a protection-check condition is indicated in the channel status word (CSW) stored as a result of the operation.

When the access to storage is inhibited by the processor, and protection applies, the protection key of the processor occupies bit positions 8-11 of the PSW. When the reference is made by a channel, and protection applies, the protection key associated with the I/O operation is used as the comparand. The protection key for an I/O operation is specified in bit positions 0-3 of the channel-address word (CAW) and is recorded in bit positions 0-3 of the channel status word (CSW) stored as a result of the I/O operation.

To use fetch protection, a virtual machine must execute the Set Storage Key (SSK) instruction referring to the data areas to be protected, with the fetch protect bit set on in the key. VM/SP subsequently:

1. Checks for a fetch protect violation in handling privileged and nonprivileged instructions.

2. Saves and restores the fetch protect bit (in the virtual storage key) when writing and recovering virtual machine pages from the paging device.

3. Checks for a fetch protection violation on a write CCW (except for spooling or console devices).

The CMS nucleus resides in a shared segment. This presents a special case for storage protection since the nucleus must be protected and still shared among many CMS users. In order to protect the CMS nucleus in the shared segment, user programs and disk-resident CMS commands run with a different key than the nucleus code.

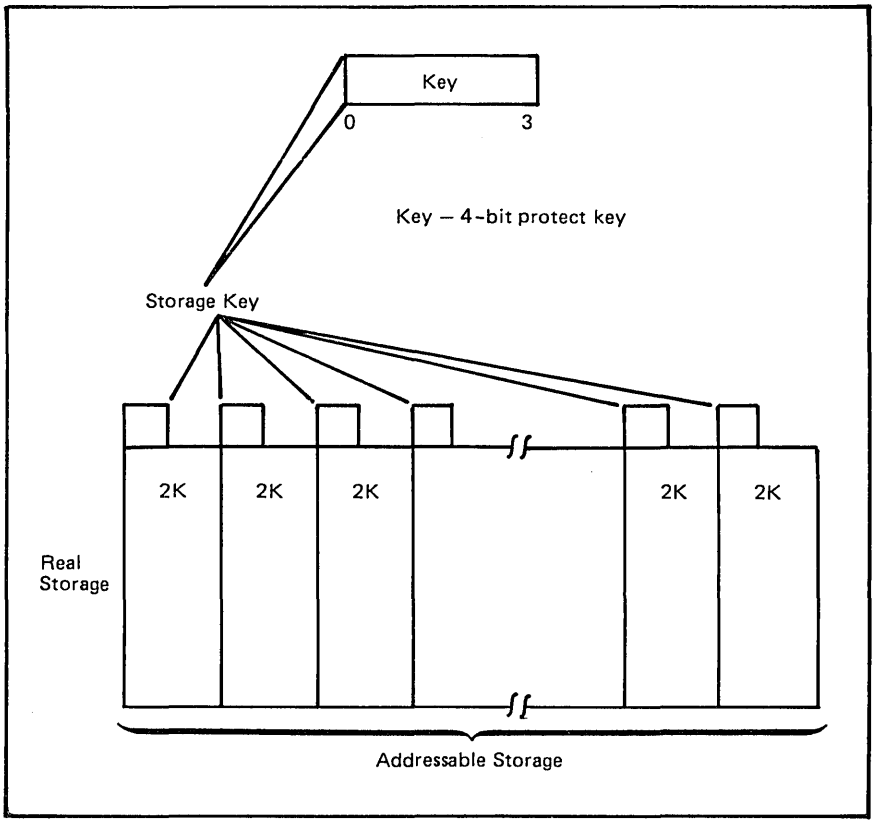



**Figure 2. 2K Storage Protection Key**

## Storage and Processor Utilization

The system operator may assign the reserved page frames option to a single virtual machine. This option, specified by the SET RESERVE command, assigns a specific amount of the storage of the real machine to the virtual machine. CP dynamically builds up a set of reserved real storage page frames for this virtual machine during its execution until the maximum number "reserved" is reached. Since the pages of other virtual machines are not allocated from this reserved set, the effect is that most of the active pages of the selected virtual machine remain in real storage.

During CP system generation, the installation may specify an option called virtual=real. With this option, the virtual machine's storage is allocated directly from real storage at the time the virtual machine logs on (if it has the VIRT=REAL option in its directory entry). All pages except page zero are allocated to the corresponding real storage locations. In order to control the real computing system, real page zero must be controlled by CP. Consequently, the real storage size must be large enough to accommodate the CP nucleus, the entire virtual=real virtual machine, and the remaining pageable storage requirements of CP and the other virtual machines.

The virtual=real option improves performance in the selected virtual machine since it removes the need for CP paging operations for the selected virtual machine. The virtual=real option is necessary whenever programs that contain dynamically modified channel programs (excepting those of OS ISAM and OS/VS TCAM Level 5) are to execute under control of CP.

During CP system generation, the installation can specify an option called the "Small CP Option". The Small CP option removes some of the normally resident CP nucleus functions that support remote CP. This effectively reduces the size of the resident CP nucleus, making more storage available for the area where virtual machine pages reside.

The Small CP option improves performance in environments where the real processor storage size is 512K bytes or less.

## *Virtual Storage Preservation*

Virtual storage preservation support is designed to preserve the contents of a virtual machine if the system operator forces the machine off the system, if VM/SP abnormally terminates it, or if VM/SP itself abnormally terminates.

The user can specify at VM/SP system generation time which virtual machines are to be saved. The contents of these virtual machines are saved in DASD space that the VM/SP system programmer must previously allocate via the NAMESYS macro. The user can force a priority for the order in which multiple virtual machines are saved; he does this through the SAVESEQ operand of the NAMESYS macro. The saved virtual machine is restored to the user via the IPL command. Normal recovery procedures or problem analysis for the saved virtual machine may then be initiated by the user. To preserve its privacy and security, the automatically saved virtual machine is made available only to previously specified users. This saved virtual machine can be loaded into either a V=R or a normal non-V=R machine.

### VMSAVE Option

Subject to certain restraints, the user can dynamically control the option to save or not to save the contents of the virtual machine (VMSAVE), and in which DASD area to save them (if there is more than one DASD area). If the user has a single DASD area defined, VMSAVE can be enabled either by the VMSAVE directory option or by the SET VMSAVE ON command. A single VMSAVE area can be designated for use by multiple virtual machines. However, the area is allocated to only one user at a time; the user who first enables VMSAVE has priority. Normal logoff, or invoking the SET VMSAVE OFF command relinquishes this VMSAVE area.

The user with multiple DASD areas allocated must issue the SET VMSAVE name command to enable the VMSAVE option. The SET VMSAVE OFF command disables the VMSAVE option. Also, to relinquish the VMSAVE area, the user may issue the SET VMSAVE OFF command, or logoff, or issue the SET VMSAVE name command specifying another area. The DASD save area can only be relinquished by the owner of the data of the save area if data is stored in it. If there is a saved system in the DASD area, the way to relinquish the area is for the owner of the saved area to logon and issue the SET VMSAVE name command for that area, then issue SET VMSAVE OFF command or LOGOFF.

The current status of the VMSAVE option (ON or OFF) can be obtained from the QUERY SET command. The QUERY VMSAVE command displays the current status of the VMSAVE option, the names of the areas allocated for the user, the page frames of each area, and the date and time that their contents were saved.

### Termination

If the VMSAVE option is enabled when conditions of termination other than normal LOGOFF occur (such as a VM/SP abend and restart), the pages of the virtual machine specified are saved in the previously allocated DASD area in the order specified at system generation time by NAMESYS macro values in DMKSNT.

### IPL

After a virtual machine termination or a VM/SP abend in which virtual machine contents were saved by the VMSAVE option, the IPL command initiated for the designated VMSAVEd system by a logged-on user brings a page image copy of a saved virtual machine into an active virtual machine, but does not give the saved virtual machine control. The copy can always be dumped; however, it may or may not be executable.

The V=R area (if active) of the real machine is preserved if the system is performing a warm start. The V=R area is cleared if the system terminates to a hard wait state or if a different V=R user logs on.

### Priority

The SAVESEQ operand of the NAMESYS macro allows the user to force a priority in the saving order of multiple virtual machines. (The NAMESYS macro is described in detail in the *VM/SP Planning Guide and Reference.*) The priority is determined by number. The lower the number, the higher the priority. If two virtual machines have the same priority, and both have the VMSAVE option enabled, they are saved in the order in which they enabled VMSAVE. A sequence of VMSAVE disable followed by a VMSAVE enable causes a virtual machine to be the last one on the chain -- that is, last among the other virtual machines that have the same SAVESEQ priority value.

If a high priority of SAVESEQ is specified for the production virtual machine, and lower or equal priorities are specified for other virtual machines, the production machine is saved first; other virtual machines are saved in the order in which the virtual machines logged onto the system.

If different values of SAVESEQ are specified for each user (the range is 0-255), the priority of saving order for each virtual machine is predictable, depending on which users are logged on when an abend occurs.

### VMSAVE Areas

The VM/SP FORMAT/ALLOCATE program must format DASD space used for VMSAVE areas before any user can store into the area. Detailed information on using the FORMAT/ALLOCATE program is contained in the *VM/SP Operator's Guide.*

### Target Areas

You can specify multiple VMSAVE target areas for a single user; you do this by including in the DMKSNT module more than one NAMESYS macro with the same USERID=operand. Different target areas are required if a user wishes to IPL a VMSAVE system and have the VMSAVE option enabled at the same time. Once the VMSAVE is enabled, the area referred to cannot be referenced by the IPL command until a recovery operation has been effected. Similarly, if a VMSAVE area currently contains a saved system, it can be released only by the user who

caused the system to be stored there. That area cannot be the VMSAVE target area referred to by a VMSAVE enable from another user until the stored system has been released.

### Overlapping Areas

The system programmer, at his option, can specify overlapping DASD areas for VMSAVE target areas through NAMESYS macro specifications. However, if two areas overlap, they must start at the same physical cylinder and page. They can end at different locations if the areas are of different lengths. Overlapping areas are useful for different environments of the same user, and they are also valid as VMSAVE target areas for different users.

Only one user can be using the area (for IPL or for a VMSAVE target area) at any one time. In addition, if one user has caused a virtual machine to be stored into an area, no other user can access that area. The user also cannot issue the SET VMSAVE command with that area as the VMSAVE target area, until the user who caused the virtual machine to be stored does the following:

- Enables VMSAVE to that area via the SET command, which effectively clears the area.
- Releases the area by issuing a SET VMSAVE command to another area, a SET VMSAVE OFF, a DEFINE STORAGE, or a normal LOGOFF process.

Only when the area has been cleared and released in this manner is it available for other users.

For overlapping target areas, the user must load a system that has the same name that it was saved under. This ensures that the page range returned with the load is the same as the one stored by VMSAVE.

Only when the complete page range specified has been saved does the area become valid and available. If an error occurs in the middle of a save operation, the area is not valid, and therefore is not retrievable.

The user cannot force a save directly. The MESSAGE command may be used to ask the operator to force the user off the system. The FORCE command causes an automatic save, assuming that VMSAVE is enabled. The user can also disconnect with a READ pending. After 15 minutes the system logs off the user, causing an automatic save if VMSAVE is enabled.

### Other Saved Systems

Systems loaded by name under VM/SP must be saved by the SAVESYS command under VM/SP. Because of control block changes, systems saved under other releases of VM/370 are not loaded properly on VM/SP. Conversely, systems saved on VM/SP will not load properly on a system that does not have this product installed.

## *Virtual Machine I/O Management*

A real disk device can be shared among multiple virtual machines. Virtual device sharing is specified in the VM/SP directory entry or by a user command. If specified by the user, an appropriate password may have to be supplied before gaining access to the virtual device. A particular virtual machine may be assigned

read-only or read/write access to a shared disk device. CP checks each virtual machine input/output operation against the parameters in the virtual machine configuration to ensure device integrity.

Virtual Reserve/Release support can be used to further enhance device integrity for data on shared minidisks. Reserve/Release operation codes are simulated on a virtual basis for minidisks, including full-extent minidisks. For details on Reserve/Release support, refer to the *VM/SP System Logic and Problem Determination Guide, Volume 1.*

The virtual machine operating system is responsible for the operation of all virtual devices associated with it. These virtual devices may be defined in the VM/SP directory entry of the virtual machine, or they may be attached to (or detached from) the virtual machine's configuration, dynamically, for the duration of the terminal session. Virtual devices may be dedicated, as when mapped to a fully equivalent real device; shared, as when mapped to a minidisk or when specified as a shared virtual device; or spooled by CP to intermediate direct access storage.

In a real machine, input/output operations are normally initiated when a problem program requests the operating system to issue a START I/O instruction to a specific device. Device error recovery is handled by the operating system. In a virtual machine, the operating system can perform these same functions, but the device address specified and the storage locations referenced are both virtual. It is the responsibility of CP to translate the virtual specifications to real.

In addition, the interrupts caused by the input/output operation are reflected to the virtual machine for its interpretation and processing. If input/output errors occur, CP records them but does not initiate error recovery operations. The virtual machine operating system must handle error recovery, but does not record the error (if SVC 76 is used).

In an attached processor environment, virtual I/O can be initiated by either processor; however, all real I/O requests must be executed by the main processor, and all I/O interrupts must be received on the main processor (the processor with I/O capability). Any I/O requests by the attached processor (the processor without I/O capability) are transferred to the main processor.

In a multiprocessor environment, both processors have real I/O capability. If either processor receives an I/O request, that processor attempts to initiate I/O operations. If none of the online paths from the executing processor to the required device are available, that processor queues the I/O request on all busy and scheduled paths to the device; both its own and the alternate paths to the device from the second processor. If there is no online path from the executing processor, that processor queues the I/O request on the first online and available path for the second processor, as well as on all busy or scheduled paths from that processor.

Input/output operations initiated by CP for its own purposes (paging and spooling), are performed directly and are not subject to translation.

Virtual machines may access data on MSS mass storage volumes using that virtual machine's standard 3330 device support. MSS cylinder faults, and associated asynchronous interruptions, are transparent to the virtual machine in this situation.

### Dedicated Channels

In most cases, the I/O devices and control units on a channel are shared among many virtual machines as minidisks and dedicated devices, and shared with CP system functions such as paging and spooling. Because of this sharing, CP has to schedule all the I/O requests to achieve a balance between virtual machines. In addition, CP must reflect the results of the subsequent I/O interruption to the appropriate storage areas of each virtual machine.

By specifying a dedicated channel (or channels) for a virtual machine via the Class B ATTACH CHANNEL command, the CP channel scheduling function is bypassed for that virtual machine. A virtual machine assigned a dedicated channel has that channel and all of its devices for its own exclusive use. CP translates the virtual storage locations specified in channel commands to real locations and performs any necessary paging operations, but does not perform any device address translations. The virtual device addresses on the dedicated channel must match the real device addresses; thus, a minidisk cannot be used.

## *Spooling Functions*

A virtual unit record device, which is mapped directly to a real unit record device, is said to be dedicated. The real device is then controlled completely by the virtual machine's operating system.

CP facilities allow multiple virtual machines to share unit record devices. Since virtual machines controlled by CMS ordinarily have modest requirements for unit record input/output devices, such device sharing is advantageous, and it is the standard mode of system operation.

Spooling operations cease if the direct access storage space assigned to spooling is depleted, and the virtual unit record devices appear in a not-ready status. The system operator or the spooling operator may make additional spooling space available by purging existing spool files or by assigning additional direct access storage space to the spooling function. The spooling operator can use the class D SPTAPE command to retrieve spool files from tape for output processing when spooling space requirements are not critical. See the description of the SPTAPE command in the *VM/SP Operator's Guide* for further information.

Specific files can be transferred from the spooled card punch or printer of a virtual machine to the card reader of the same or another virtual machine. Files transferred between virtual unit record devices by the spooling routines are not physically punched or printed. With this method, files can be made available to multiple virtual machines, or to different operating systems executing at different times in the same virtual machine.

Files may also be spooled to remote stations via the Remote Spooling Communications Subsystem (RSCS), a program product of VM/SP.

CP spooling includes many desirable options for the virtual machine user and the real machine operator. These options include printing multiple copies of a single spool file, backspacing any number of printer pages, and defining spooling classes for the scheduling of real output. Each output spool file has, associated with it, a 136-byte area known as the spool file tag. The information contained in this area and its syntax are determined by the originator and receiver of the file. For example, whenever an output spool file is destined for transmission to a remote location

via the Remote Spooling Communications Subsystem, RSCS expects to find the destination identification in the file tag. Tag data is set, changed, and queried using the CP TAG command.

It is possible to spool terminal input and output. All data sent to the terminal, whether it be from the virtual machine, the control program or the virtual machine operator, can be spooled. Spooling is particularly desirable when a virtual machine is run with its console disconnected. Console spooling is usually started via the command

```
SPOOL CONSOLE START
```

An exception to this is when a system operator logs on using a graphics device. In this instance, console spooling is automatically started and continues in effect even if the system operator should disconnect from the graphics device and log on to a nongraphic device. In order to stop automatic console spooling, the system operator must issue the command

```
SPOOL CONSOLE STOP
```

## *Spool File Recovery*

If the system should suffer an abnormal termination, there are three degrees of recovery for the system spool files; warm start (WARM), checkpoint start (CKPT), and force start (FORCE). Warm start is automatically invoked if SET DUMP AUTO is in effect. Otherwise, the choice of recovery method is selected when the following message is issued:

```
hh:mm:ss START ((COLD|WARM|CKPT|FORCE)(DRAIN))|(SHUTDOWN):
```

Note that a cold (COLD) start does not recover any spool files.

### Warm Start

After a system failure, the warm start procedure copies spool file, accounting, and system message data to the warm start area on the IPLed system residence volume. When the system is reloaded, this information is retrieved and the spool file chains and other system data are restored to their original status. If the warm start procedure cannot be implemented because certain required areas of storage are invalid, the operator is notified to take other recovery procedures.

### Checkpoint Start

Any new or revised status of spool file blocks, spooling devices, and spool hold queue blocks is dynamically copied to the checkpoint area on the IPLed system residence volume as it occurs. When a checkpoint (CKPT) start is requested, this is the information that is used to recreate the spool file chains. It differs from warm start data in that only spool file data is restored; accounting and system messages information is not recovered. Also, the order of spool files on any particular restored chain is not the original sequence but a random one.

### Force Start

A force start is required when checkpoint start encounters I/O errors while reading files, or invalid data. The procedure is the same as for checkpoint start except that unreadable or invalid files are bypassed.

## *CP Commands*

The CP commands allow you to control the virtual machine from the terminal, much as an operator controls a real machine. Virtual machine execution can be stopped at any time by using the 3066 terminal's attention key or the 3270 terminal's ENTER or PA1 key. Execution can be restarted by entering the appropriate CP command. External, attention, and device ready interrupts can be simulated on the virtual machine. Virtual storage and virtual machine registers can be inspected and modified, as can status words such as the PSW and the CSW. Extensive trace facilities are provided for the virtual machine, as well as a single-instruction mode. Commands are available to invoke the spooling and disk sharing functions of CP.

CP commands are classified by privilege classes. The VM/SP directory entry for each user assigns one or more privilege classes. The classes are primary system operator (class A), system resource operator (class B), system programmer (class C), spooling operator (class D), system analyst (class E), service representative (class F), and general user (class G). Commands in the system analyst class may be used to inspect real storage locations, but may not be used to make modifications to real storage. Commands in the operator class provide real resource control capabilities. System operator commands include all commands related to virtual machine performance options, such as assigning a set of reserved page frames to a selected virtual machine. For descriptions of all the CP commands, see the *VM/SP CP Command Reference for General Users* and the *VM/SP Operator's Guide.*

## Program States

When instructions in the Control Program are being executed, the real computer is in the supervisor state; at all other times, when running virtual machines, the real computer is in the problem state. Therefore, privileged instructions cannot be executed by the virtual machine. Programs running on a virtual machine can issue privileged instructions; but such an instruction either (1) causes an interruption that is handled by the Control Program, or (2) is intercepted and handled by the processor, if the virtual machine assist feature or VM/370 Extended Control-Program Support is enabled and supports that instruction. CP examines the operating status of the virtual machine PSW. If the virtual machine indicates that it is functioning in supervisor mode, the privileged instruction is simulated according to its type. If the virtual machine is in problem mode, the privileged interrupt is reflected to the virtual machine.

Only the Control Program may operate in the supervisor state on the real machine. All programs other than CP operate in the problem state on the real machine. All user interrupts, including those caused by attempted privileged operations, are handled by either the control program or the processor (if the virtual machine assist feature or VM/370 Extended Control-Program Support is available). Only those interrupts that the user program would expect from a real machine are reflected to it. A problem program executes on the virtual machine in a manner identical to its execution on a real System/370 processor, as long as the problem program does not violate the CP restrictions. CP restrictions are documented in the *VM/SP Planning Guide and Reference.*

# Using Processor Resources

CP allocates the processor resource to virtual machines according to their operating characteristics, priority, and the system resources available.

Virtual machines are dynamically categorized at the end of each time slice as interactive or noninteractive, depending upon the frequency of operations to or from either the virtual system console or a terminal controlled by the virtual machine.

Virtual machines are dispatched from one of three queues, called Queue 1, Queue 2, and Queue 3. In order to be dispatched from a queue, a virtual machine must be considered executable (that is, not waiting for some activity or for some other system resource). Virtual machines are not considered dispatchable if the virtual machine:

- Enters a virtual wait state after an I/O operation has begun.
- Is waiting for a page frame of real storage.
- Is waiting for an I/O operation to be translated by CP and started.
- Is waiting for CP to simulate its privileged instructions.
- Is waiting for a CP console function to be performed.

## Queue 1

Virtual machines in Queue 1 (Q1) are considered conversational or interactive users, and enter this queue when an interrupt from a terminal is reflected to the virtual machine. The Q1 virtual machines are ordered by their deadline priorities in the dispatch list. A deadline priority is a value calculated by the fair share scheduler every time a user is dropped from a queue (queue drop time). This value is based on paging activity, processor usage, the load on the system, and user priority. Deadline priority is used to determine when the user receives his next time slice.

A particular virtual machine's deadline priority for Q1 is better (earlier) than its corresponding priority for Q2. The deadline priorities for all Q1 virtual machines are not necessarily better than the deadline priorities for all Q2 virtual machines.

Virtual machines are dropped from Q1 when they complete their time slice of processor usage, and are placed in an "eligible list". Virtual machines entering CP command mode are also dropped from Q1.

## Queue 2

Virtual machines are selected to enter Q2 from a list of eligible virtual machines (the eligible list). The ordering of virtual machines on the eligible list and the dispatch list is determined on the basis of each virtual machine's deadline priority.

There are two lists of virtual machines in Q2; those in the eligible list and those in the dispatch list. Both lists are sorted by deadline priority. A particular deadline priority depends on many factors:

- The time-of-day the virtual machine last dropped from the dispatch list
- The virtual machine's user priority

- The current load and number of virtual machines on the system
- The current resource utilization of the virtual machine

A virtual machine enters Q2 only if its working set size is not greater than the number of real page frames available for allocation at the time. The working set of a virtual machine is calculated and saved each time a user is dropped from Q2. The working set size is a function of the number of virtual pages referred to by the virtual machine during its stay in Q2, and the number of its virtual pages that are resident in real storage at the time it is dropped from the queue.

If the calculated working set of the highest priority virtual machine in the eligible list is greater than the number of page frames available for allocation, CP continues to search the eligible list, in deadline priority order, for a virtual machine whose working set does not exceed the number of available page frames.

When a virtual machine completes its time slice of processor usage, it is dropped from Q2 and placed in the eligible list according to its deadline priority. When a virtual machine in Q2 enters CP command mode, it is removed from Q2.

To leave CP mode and return his virtual machine to the eligible list for Q2, a user can issue a CP command that transfers control to the virtual machine operating system for execution (for example, BEGIN, IPL, EXTERNAL, and RESTART).

Virtual machines in Q2 are considered to be noninteractive. In CP, interactive virtual machines (those in Q1), if any, are normally considered for dispatching before noninteractive virtual machines (Q2). This means that CMS users entering commands that do not involve disk or tape I/O operations should get fast responses from the VM/SP system even with a large number of active virtual machines. All virtual machines (Q1 and Q2) on the dispatch list are ordered by their deadline priority. There can be many instances where some virtual machines in Q2 are considered for dispatching before virtual machines in Q1 because of their user priority, current resource utilization level, or for other reasons.

## Deadline Priority

The deadline priority is calculated at queue drop time by:

deadline priority = TOD + user bias factor

*where:*

TOD
: is the current time of day.

User bias factor
: is the user bias ratio * Q2 delay factor

User bias ratio
: is less than 1, equal to 1, or greater than 1, depending on the whether the particular virtual machine is currently receiving less than, equal to, or more than its specified amount of resources.

Q2 delay factor
: is calculated dynamically based on configuration and load, and is the average elapsed time required by a virtual machine to receive an amount of processor time equal to one Q2 time slice.

For Q1 virtual machines, the scaled bias is divided by 8 (since the Q1 processor usage time slice is 1/8th the Q2 time slice). The difference between scheduling a virtual machine in Q1 instead of Q2 is that it receives 1/8th the amount of processor, 8 times as often. Operating constantly in either queue, a virtual machine should receive the same amount of processor resources over an extended period of time. The only preference given Q1 virtual machines is when they are being moved from the eligible list to the dispatch list. They are moved ahead of Q2 virtual machines with the same or even slightly better deadline priorities.

## Queue 3

Q3 is an extension of Q2 scheduling. It helps to distinguish between non-interactive virtual machines and those that are frequently switching back and forth between Q2 and Q1. Virtual machines that have cycled through at least eight consecutive Q2 processor time slices without a Q1 interaction are labeled Q3. Q3 virtual machines are kept in the same lists (or queues) as Q2 virtual machines and for most purposes are treated identically. The differences between Q2 and Q3 virtual machines are reflected in their deadline priority calculations and the amounts of such processor time they are allowed in queue. Q3 virtual machines are allowed eight consecutive Q2 processor time slices before they are dropped from queue. Because of the eight-fold increase in processor time allowed each time in queue, the scaled bias is multiplied by eight before adding to the current time-of-day to form the deadline priority. Q3 virtual machines should receive eight times as much processor time each time in queue as Q2 virtual machines, but only 1/8th as often.

To reiterate the Q1/Q2 statement, which is also true for Q2/Q3: Operating constantly in any queue, a virtual machine should receive the same amount of processor resources over an extended period of elapsed time. This does not necessarily mean that a virtual machine performs the same when operating in Q3 mode as when operating in standard Q2 mode. An amount of overhead (roughly proportional to the small number of resident pages) is used for each virtual machine when it drops from queue. When operating in Q3 mode, a virtual machine may perform much better than in normal Q2 mode because it is undergoing fewer queue drops. For some very large virtual storage programs, the total processor resources used has been cut in half by operating in Q3 mode as compared to standard Q2 mode.

### CMS BLIP Facility

The CMS BLIP facility causes CMS to perform a write operation to the terminal after every 2 seconds of virtual processor use. This feature effectively cancels Queue 3 use for normal, connected CMS virtual machines, regardless of what types of programs they are running. The CMS BLIP facility can be turned off with the CMS SET BLIP OFF command or it can be disabled with the CP SET TIMER OFF command.

# Interruption Handling

## I/O Interrupts

Input/output interrupts from completed I/O operations initiate various completion routines and the scheduling of further I/O requests. The I/O interrupt handling routine also gathers device sense information.

## Missing Interrupt Handler

An I/O operation, such as a minidisk operation or a paging operation, that does not complete in a specified time period causes a missing interrupt condition. An incomplete minidisk operation can lock out a virtual machine user or an incomplete paging I/O operation can degrade the performance of the system. The missing interrupt handler detects incomplete I/O conditions by monitoring I/O activity and, in addition, it takes action to correct incomplete I/O conditions without operator intervention. The missing interrupt handler, therefore, is designed to improve the availability of the system by preventing user lockout and system degradation.

The missing interrupt handler scans the real device blocks (RDEVBLOKs) at specified time intervals. If the device is busy (RDEVBUZY flag is on) a bit (RDEVMID) is set that indicates a possible missing interrupt condition. The first level interrupt handler, DMKIOT, resets RDEVBUZY and RDEVMID when the device causes an interrupt at the completion of an I/O operation. Therefore, if RDEVMID is on at the end of the next time interval, a missing interrupt condition exists.

The installation may use the default time interval for each distinct device category or may specify a time value. For example, if the default time interval value of ten minutes for tape devices is not appropriate for an installation's configuration, the installation may change this value. See "Default Time Interval Values" and "Changing the Time Interval" for a list of the default time interval values and how you can change these values.

### *Using the Missing Interrupt Handler*

To use the Missing Interrupt Handler, DMKDID must be included in the loadlist during system generation. MIH can be set on either by including it as an option in the directory or by issuing the SET command. The default is MIH OFF. With MIH is on, when a missing interrupt is detected, CP simulates the interrupt. With MIH off, when a missing interrupt is detected, message DMKDID546I is issued but CP does not simulate the interrupt. If DMKDID is deleted from the load list during system generation, support for the Missing Interrupt Handler is removed and no messages are written to notify the operator of a missing interrupt.

If you want to change the interval time value, you must include the optional macro SYSMIH in the system control file (DMKSYS). You must place this macro before the SYSLOCS macro.

When a missing interrupt occurs, the control program attempts to correct the condition and issues a message that either:

- The condition is cleared

- or -

- The condition is pending

This message warns the system operator or system programmer that a problem may exist. The system operator or the system programmer can reset the hardware and schedule maintenance for the device that caused the missing interrupt condition. If the same device class caused frequent interruptions, the system programmer may want to set a larger time interval for that particular device class.

The class G SET command can be used to turn MIH on and off. Use either

SET MIH ON or SET MIH OFF

To determine the status of MIH use

QUERY SET

The system responds either

MIH ON or MIH OFF

### *Devices Monitored*

Each device group has an expected time interval during which an I/O operation should be completed. This interval varies widely among devices. Therefore, the missing interrupt handler provides a means to specify a time interval for the following distinct categories of I/O devices:

- Count-key-data devices (CLASDASD) and FB-512 devices (CLASFBA)
- Tape devices (CLASTAPE)
- Graphic devices (CLASGRAF) except TYP1053 and TYP328X
- Unit record devices (CLASURI and CLASURO) except TYP3800 and TYP3289E
- Miscellaneous devices (MISC) include: Mass storage system (MSS) devices (specified at system generation as CLASSPEC TYP3851, and CLASDASD FEATURE=VIRTUAL or FEATURE=SYSVIRT), graphics devices TYP1053 and TYP328X, and UR output devices TYP3800 and TYP3289E.

**Note:** The missing interrupt handler does not support terminal devices, remote graphic devices, SNA devices, pass-through virtual machine (logical) devices, and special class devices (with the exception of MSS).

### *Default Time Interval Values*

Default time interval values are assembled in DMKSYS. The following table gives the default time interval values for the devices monitored:

| Device Class | Class Parameter | Default Time Interval |
|---|---|---|
| CLASDASD or CLASFBA | DASD | 15 seconds |
| CLASGRAF | GRAF | 30 seconds |
| CLASTAPE | TAPE | 10 minutes |
| CLASURI/CLASURO | UR | 1 minute |
| MISCELLANEOUS | MISC | 12 minutes |

An installation may want to change the default time intervals because of their particular configuration. For example, an installation that generates a large number of devices might want to set the time interval value to a larger number to prevent frequent timer interruptions.

### *Changing the Time Interval*

The system programmer or the system operator can change the time interval in the following ways:

- Regenerate the system and, using the SYSMIH macro, specify a time interval value in the system control file (DMKSYS) for the specific device class to be changed. Specify the time interval value in minutes and seconds:

  ```
  SYSMIH GRAF=00:15,UR=00:00,TAPE=05:00
  ```

  This example changes the time interval for graphic devices from the default value of thirty seconds to fifteen seconds. In this example, no further monitoring takes place for unit record devices since the user specified a time value of zero for that class. In addition, the example changes the time interval value for tape devices from ten minutes to five minutes. This example does not change the time interval value for DASD and MISC devices. If you do not specify a device class, or if you do not include the SYSMIH macro in DMKSYS, the missing interrupt handler uses the default value for that class.

- To change the value specified in DMKSYS for a particular device class, issue the class B CP command specifying the new time interval value for that class in minutes and seconds:

  ```
  SET MITIME GRAF 00:10
  ```

  This example changes the time interval for graphic devices to ten seconds. This change is in effect until the system is reinitialized, or until a class B user issues another SET MITIME command. If the user specifies a time value of zero for a specific device class, no further monitoring takes place for that device class.

  **Note:** If you set the time interval for a device class below its default value, be careful not to shorten the time interval too much. This may cause unnecessary missing interruption handler processing for devices that are functioning properly.

- To set all time values to zero and to prevent any monitoring for missing interrupts for any devices, issue the class B CP command:

  ```
  SET MITIME OFF
  ```

Monitoring for missing interrupts does not take place until the system is reinitialized, or until the class B user issues another SET MITIME command.

### *Determining Time Interval Settings*

The class B user can determine the current missing interrupt handler time intervals by issuing the following CP command:

```
QUERY MITIME
```

The system issues:

- The time interval setting for each device group in minutes and seconds
- The response MITIME OFF
- An error message if the user specified an invalid parameter.
- The response that the missing interrupt handler is not available if DMKDID is not in the loadlist during system generation.

### *Diagnostic Aids*

Missing interrupt handler support provides aids so that the system programmer can determine the frequency and status of interrupts and also know when he has made an error in using the support. Diagnostic aids available when using the missing interrupt handler include:

- System messages
- Macro notes
- VM/SP system's error recording area
- Trace table

### *System Messages*

Messages inform the system operator when a missing interrupt occurs and indicate if the condition has been cleared or if the interrupt is still pending. Other messages indicate that the module DMKDID is not in the loadlist or that the user specified an invalid parameter on the QUERY or SET MITIME command. See *VM/SP System Messages and Codes* for a complete discussion of messages that the missing interrupt handler issues.

The system programmer can use message information to increase the availability of the system. If a particular device class causes frequent interrupts even if the system clears the condition, the system programmer may want to change the time interval. Changing the time interval prevents the overhead of frequent timer interrupts, frequent trips through the detector routine, and rescheduling of timer request queues. On the other hand, if the control program did not clear the condition, the messages make the system programmer or system operator aware of the condition and one of them can reset the hardware either physically or using CP commands.

Macro notes (MNOTES) inform the user that SYSMIH is not present in DMKSYS or that the user specified an invalid time value in the SYSMIH macro. The system uses the default interval time values and informs the user.

### *System's Error Recording Area*

Whether or not CP succeeds in correcting a missing interrupt situation, it creates a record of the event in the system's error recording area (LOGREC).

### *Trace Table*

CP also traces the simulated interrupt and records it as trace table entry X'19'. Refer to Figure 67 on page 507, "CP Trace Table Entries", for the format of the entry. The system programmer uses the trace table to determine the events that preceded a CP system failure.

## Program Interrupt

Program interrupts can occur in two states. If the processor is in supervisor state, the interrupt indicates a system failure in the CP nucleus and causes the system to abnormally terminate. If the processor is in problem state, a virtual machine is executing. CP takes control to perform any required paging operations to satisfy the exception, or to simulate the instruction. The fault is transparent to the virtual machine execution. Any other program interrupt is a result of the virtual machine processing and is reflected to the machine for handling.

## Machine Check Interrupt

When a machine check occurs, the CP Recovery Management Support (RMS) gains control to save data associated with the failure for the Field Engineer. RMS analyzes the failure to determine the extent of damage.

Damage assessment results in one of the following actions being taken:

- System termination (CP disabled wait state)
- Attached processor disabled (system continues in uniprocessor mode)
- One processor of a multiprocessor configuration disabled (system continues in uniprocessor mode)
- One or more failing channels disabled (system continues in same mode as at time of the error)
- Selective virtual user termination
- Selective virtual machine reset
- Refreshing of damaged information with no effect on system configuration
- Refreshing of damaged information with the defective storage page removed from further system use
- Error recording only for certain soft machine checks

The system operator is informed of all actions taken by the RMS routines. When a machine check occurs during VM/SP startup (before the system is sufficiently initialized to permit RMS to operate successfully), the processor goes into a disabled wait state and places a completion code of X'00B' in the leftmost bytes of the current PSW.

## SVC Interrupt

When an SVC interrupt occurs, the SVC interrupt routine is entered. If the machine is in problem mode, the type of interrupt (if it is other than an SVC 76 or ADSTOP SVC) is reflected to the pseudo-supervisor (that is, the supervisor operating in the user's virtual machine). Control is transferred to the appropriate interrupt handler for ADSTOP SVCs and all SVC 76s.

If the machine is in supervisor mode, the SVC interrupt code is determined, and a branch is taken to the appropriate SVC interrupt handler.

## External Interrupt

If a timer interrupt occurs, CP processes it according to type. The interval timer indicates time slice end for the running user. The clock comparator indicates that a specified timer event occurred, such as midnight, scheduled shutdown, or user event reached.

The external console interrupt invokes CP processing to switch from the 3210 or 3215 to an alternate operator's console.

A service signal is a class 24 external interrupt that is generated when either a logical device or the Maintenance and Service Support Facility (MSSF) signals completion of an operation initiated by a program (in the case of the logical device DIAGNOSE X'7C') or CP, (in the case of the MSSFCALL DIAGNOSE X'80').

See the expanded descriptions of DIAGNOSE codes X'7C' and X'80' in "Part 1. Control Program (CP)". Also refer to *IBM System/370 Principles of Operation,* GA22-7000 for a general description of external interrupts.

## Synchronous Interrupts in an Attached Processor or Multiprocessor System

Generally, when synchronous interrupts (such as program and SVC interrupts) occur in an attached processor or multiprocessor system, the processing of the interrupt can proceed without the global system lock for mainline, nonerror paths. Otherwise, the global system lock is required. If the global system lock is needed and it is already in use, the processing of the interrupt is deferred until the global system lock is available. In this case, the interrupted processor attempts to run another user.

## Real I/O Interrupts

In an attached processor configuration, only the main processor can receive real I/O interrupts. To ensure this, the channel masks in control register 2 on the main processor are initialized to ones to enable interruptions from any available channel. On the attached processor, the channel masks in control register 2 are initialized to zeros. In a multiprocessor configuration, both processors can receive real I/O interruptions. The channel masks in control register 2 on both processors are initialized to ones to enable interruptions from any available channel.

# Performance Guidelines

## General Information

The performance characteristics of an operating system, when it is run in a virtual machine environment, are difficult to predict. This unpredictability is a result of several factors:

- The System/370 model used.
- The total number of virtual machines executing.
- The type of work being done by each virtual machine.
- The speed, capacity, and number of the paging devices.
- The amount of fixed head paging storage (drum, 3340, 3344, 3350, 3380)
- The amount of real storage available.
- The degree of channel and control unit contention, as well as arm contention, affecting the paging device.
- The type and number of VM/SP performance options in use by one or more virtual machines.
- The degree of MSS 3330 volume use.
- The order in which devices are selected for preferred paging and spooling.

Performance of any virtual machine may be improved by the choice of hardware, operating system, and VM/SP options. The topics discussed in this section address:

1. The performance options available in VM/SP to improve the performance of a particular virtual machine.
2. The system options and operational characteristics of operating systems running in virtual machines that affect their execution in the virtual machine environment.

The performance of a specific virtual machine may never equal that of the same operating system running standalone on the same System/370, but the total throughput obtained in the virtual machine environment may equal or better that obtained on a real machine.

When executing in a virtual machine, any function that cannot be performed wholly by the hardware causes some degree of degradation in the virtual machine's performance. As the control program for the real machine, CP initially processes all real interrupts. A virtual machine operating system's instructions are always executed in *problem* state. Any privileged instruction issued by the virtual machine causes a real privileged instruction exception interruption. The amount of work to be done by CP to analyze and handle a virtual machine-initiated interrupt depends upon the type and complexity of the interrupt.

The simulation effort required of CP may be trivial, as for a supervisor call (SVC) interrupt (which is generally reflected back to the virtual machine), or may be more complex, as in the case of a Start I/O (SIO) interrupt, which initiates extensive CP processing.

When planning for the virtual machine environment, consideration should be given to the number and type of privileged instructions to be executed by the virtual machines. Any reduction in the number of privileged instructions issued by the virtual machine's operating system reduces the amount of extra work CP must do to support the machine.

## Virtual Machine I/O

To support I/O processing in a virtual machine, CP must translate all virtual machine channel command word (CCW) sequences to refer to real storage and real devices and, in the case of minidisks, real cylinders. When a virtual machine issues an SIO, CP must:

1. Intercept the virtual machine SIO interrupt.
2. Allocate real storage space to hold the real CCW list to be created.
3. Translate the virtual data addresses to real data addresses.
4. Translate the virtual device addresses referred to in the virtual CCWs to real device addresses.
5. Page into real storage and lock, for the duration of the I/O operation, all virtual storage pages required to support the I/O operation.
6. Generate a new CCW sequence building a Channel Indirect Data Address list if the real storage locations cross page boundaries.
7. If the real device is a 3330V, append an MSS cylinder fault prefix to the CCW prefix to prevent the channel from doing channel command retry.
8. Schedule the I/O request.
9. Present the SIO condition code to the virtual machine.
10. Recognize an MSS cylinder fault, queue the I/O request, and reschedule the request when the subsequent interruption is received (indicating staging is complete).
11. Intercept, retranslate, and present the channel end and device end interrupts to the appropriate virtual machine, where they must then be processed by the virtual machine operating system.

CP's handling of SIOs for virtual machines can be one of the most significant causes of reduced performance in virtual machines.

The number of SIO operations required by a virtual machine can be significantly reduced in several ways:

- Use of large blocking factors (up to 4096 bytes) for user data sets to reduce the total number of SIOs needed.

- Use of preallocated data sets.
- Use of virtual machine operating system options (such as chained scheduling in OS) that reduce the number of SIO instructions.
- Substitution of a faster resource (virtual storage) for I/O operations, by building small temporary data sets in virtual storage rather than using an I/O device.

Frequently, there can be a performance gain when CP paging is substituted for virtual machine I/O operations. The performance of an operating system such as OS can be improved by specifying as resident as many frequently used OS functions (transient subroutines, ISAM indexes, and so forth) as are possible. In this way, paging I/O is substituted for virtual machine-initiated I/O. In this case, the only work to be done by CP is to place into real storage the page that contains the desired routine or data.

Three CP performance options are available to reduce the CP overhead associated with virtual machine I/O instructions or other privileged instructions used by the virtual machine's I/O Supervisor:

1. The virtual=real option removes the need for CP to perform storage reference translation and paging before each I/O operation for a specific virtual machine.
2. The virtual machine assist feature reduces the real supervisor state time used by VM/SP. For a detailed description of the feature, see "Virtual Machine Assist Feature" later in this section. For a list of processors on which the feature is available, see the *VM/SP Planning Guide and Reference.*
3. VM/370 Extended Control-Program Support (ECPS) further reduces the real supervisor state time used by VM/SP. For a detailed description of ECPS, see "VM/370 Extended Control-Program Support (ECPS)" later in this section. For a list of processors on which ECPS is available, see the *VM/SP Planning Guide and Reference.*

Assignment and use of these options is discussed in "VM/SP Performance Options".

## Paging Considerations

When virtual machines refer to virtual storage addresses that are not currently in real storage, they cause a paging exception and the associated CP paging activity.

The addressing characteristics of programs executing in virtual storage have a significant effect on the number of page exceptions experienced by that virtual machine. Routines that have widely scattered storage reference tend to increase the paging load of a particular virtual machine. When possible, modules of code that are dependent upon each other should be located in the same page. Reference tables, constants, and literals should also be located near the routines that use them. Exception or error routines that are infrequently used should not be placed within main routines, but located elsewhere.

When an available page of virtual storage contains only reenterable code, paging activity can be reduced, since the page, although referred to, is never changed, and

thus does not cause a write operation to the paging device. The first copy of that page is written on the paging device when that frame is needed for some other more active page. Only inactive pages that have changed must be paged out.

Virtual machines that reduce their paging activity by controlling their use of addressable space improve resource management for that virtual machine, the VM/SP system, and all other virtual machines. The total paging load that must be handled by CP is reduced, and more time is available for productive virtual machine use.

Additional dynamic paging storage may be gained by controlling free storage allocation. The amount of free storage allocated at VM/SP initialization time can be controlled by the installation. When the System is being generated, the FREE operand of the SYSCOR macro statement may be used to specify the number of free storage pages to be allocated at system load time.

If, at IPL time, the amount of storage that these pages represent is greater than 25 percent of the VM/SP storage size (not including the V=R area, if any), a default number of pages is used. The default value is 3 pages for the first 256K bytes of storage plus 1 page for each additional 64K bytes (not including the V=R size, if any).

The SYSCOR macro definition can be found in *VM/SP Planning Guide and Reference.*

CP provides three performance options, locked pages, reserved page frames, and a virtual=real area, to reduce the paging requirements of virtual machines. Generally, these facilities require some dedication of real storage to the chosen virtual machine and, therefore, improve its performance at the expense of other virtual machines.

### *Locked Pages Option*

The LOCK command, which is available to the system operator (with privilege class A), can be used to permanently fix or lock specific pages of virtual storage into real storage. In so doing, all paging I/O for these page frames is eliminated.

Since this facility reduces total real storage resources (real page frames) that are available to support other virtual machines, only frequently used pages should be locked into real storage. Since page zero (the first 4096 bytes) of a virtual machine storage is referred to and changed frequently (for example, whenever a virtual machine interrupt occurs or when a CSW is stored), it should be the first page of a particular virtual machine that an installation considers locking. The virtual machine interrupt handler pages might also be considered good candidates for locking.

Other pages to be locked depend upon the work being done by the particular virtual machine and its usage of virtual storage.

The normal CP paging mechanism selects unreferenced page frames in real storage for replacement by active pages. Page frames belonging to inactive virtual machines are all selected eventually and paged out if the real storage frames are needed to support active virtual machine pages.

When virtual machine activity is initiated on an infrequent or irregular basis, such as from a remote terminal in a teleprocessing inquiry system, some or all of its vir-

tual storage may have been paged out before the time the virtual machine begins processing. Some pages then have to be paged in so that the virtual machine can respond to the teleprocessing request compared with running the same teleprocessing program on a real machine. This paging activity may cause an increase in the time required to respond to the request compared with running the teleprocessing program on a real machine. Further response time is variable, depending upon the number of paging operations that must occur.

Locking specific pages of the virtual machine's program into real storage may ease this problem, but it is not always easy nor possible to identify which specific pages will always be required.

Once a page is locked, it remains locked until either the user logs off or the system operator (privilege class A) issues the UNLOCK command for that page. If the "locked pages" option is in effect and the user loads his system again (via IPL) or loads another system, the locked pages are refreshed and the virtual machine's locked pages are unlocked by the system. The SYSTEM CLEAR command, when invoked, clears virtual machine storage, including the user's locked pages.

**Note:** In a system generated for attached processor or multiprocessor operation, no shared pages are locked. If the system operator attempts to lock a shared page or an address range containing one or more shared pages, he receives the message

```
DMKCPV165I PAGE (hexloc) NOT LOCKED, SHARED PAGE
```

for each of the shared pages within the range.

### *Reserved Page Frames Option*

A more flexible approach than locked pages is the reserved page frames option. This option provides a specified virtual machine with an essentially private set of real page frames, the number of frames being designated by the system operator when he issues the CP SET RESERVE command. Pages are not locked into these frames. They can be paged out, but only for other active pages of the same virtual machine. When a temporarily inactive virtual machine having this option is reactivated, these page frames are immediately available. If the program code or data required to satisfy the request was in real storage at the time the virtual machine became inactive, no paging activity is required for the virtual machine to respond.

This option is usually more efficient than locked pages in that the pages that remain in real storage are those pages with the greatest amount of activity at that moment, as determined automatically by the system. Although multiple virtual machines may use the LOCK option, only one virtual machine at a time may have the reserved page frames option active. Assignment of this option is discussed further in "VM/SP Performance Options".

The reserved page frames option provides performance that is generally consistent from run to run with regard to paging activity. This can be especially valuable for production-oriented virtual machines with critical schedules, or those running teleprocessing applications where response times must be kept as short as possible.

### *Virtual=Real Option*

The VM/SP virtual=real option eliminates CP paging for the selected virtual machine. All pages of virtual machine storage, except page zero, are locked in the real storage locations they would use on a real computer. CP controls real page

zero, but the remainder of the CP nucleus is relocated and placed beyond the virtual=real machine in real storage. This option is discussed in more detail in "VM/SP Performance Options".

Since the entire address space required by the virtual machine is locked, these page frames are not available for use by other virtual machines except when the virtual=real machine is not logged on. This option often increases the paging activity for other virtual machine users, and in some cases for VM/SP. (Paging activity on the system may increase substantially, since all other virtual machine storage requirements must be managed with fewer remaining real page frames.)

The virtual=real option may be desirable or mandatory in certain situations. The virtual=real option is desirable when running a virtual machine operating system (like DOS/VS or OS/VS) that performs paging of its own because the possibility of double paging is eliminated. The option must be used to allow programs that execute self-modifying channel programs or have a certain degree of hardware timing dependencies to run under VM/SP.

## VM/SP Performance Options

VM/SP provides a number of options an installation may use to improve the performance of virtual machines and VM/SP. Several options improve the performance of installation specified virtual machines; other options improve the performance of all virtual machines and VM/SP. The options, described in the following discussion are:

- Favored execution
- User priority
- Reserved page frames
- Virtual=real
- Affinity
- Multiple shadow table support
- Shadow table bypass
- Single processor mode
- Dynamic SCP transition to or from native mode
- Queue drop elimination
- Virtual machine assist
- Extended Control-Program Support

Specifying a performance option may mean making a performance trade-off; improving the performance of one virtual machine at the expense of VM/SP and other virtual machines. For example, after an operator specifies favored execution for a virtual machine, that virtual machine receives more processor time than other virtual machines. Therefore, before specifying any performance option, identify the option's performance trade-offs and assess their impact on system performance.

The favored execution option and user priority option both alter the normal scheduler algorithm. The user priority option tends to take precedence over the favored execution option even when you specify a percentage. For example, suppose virtual machine A has favored execution nn% specified and has been given a low priority, while virtual machine B has been given a higher priority. Virtual machine A may not get the actual percentage of the CPU that was specified with the favored option.

### *Favored Execution*

The favored execution options allow an installation to modify the normal CP deadline priority calculations in the fair share scheduler to force the system to devote more of its processor resources to a given virtual machine than would ordinarily be the case. The options provided are:

- The basic favored execution option
- The favored execution percentage option

The basic favored execution option means that the virtual machine so designated is to remain in the dispatch list at all times, unless it becomes nonexecutable. When the virtual machine is executable, it is to be placed in the dispatchable list at its normal priority position. However, any active virtual machine represents either an explicit or implicit commitment of main storage. An explicit storage commitment can be specified by either the virtual=real option or the reserved page frames option. An implicit commitment exists if neither of these options is specified, and the scheduler recomputes the virtual machine's projected work-set at what it would normally have been at queue-drop time. Multiple virtual machines can have the basic favored execution option set. However, if their combined main storage requirements exceed the system's capacity, performance can suffer because of thrashing; the system can do little useful work because of excessive paging.

If the favored task is highly compute bound and must compete for the processor with many other tasks of the same type, an installation should define the processor allocation to be made. In this case, the favored execution percentage option can be selected. This option specifies that the selected virtual machine, in addition to remaining in queue, is requesting a specified minimum percentage (from 1 to 100 percent) of the total processor time, if it can use it. If a virtual machine requests 100 percent of the processor time, CP keeps that virtual machine at the top of the dispatch list. This ensures that the virtual machine always has first priority when CP dispatches a virtual machine to the processor. To select the favored execution option, specify the FAVORED operand on the class A, B, or F SET command. After the option is invoked, VM/SP provides processor time for the selected virtual machine as follows:

1. The in-queue time slice is multiplied by the specified percentage to arrive at the virtual machine's requested processor time.

2. The scheduler attempts to place the virtual machine, when it is executable, at the top of the dispatchable list until it has obtained its requested processor time.

3. If the virtual machine obtains its requested processor time before the end of its in-queue time slice, it is placed in the dispatchable list according to its calculated dispatching priority.

4. In either case (2 or 3), at the end of the in-queue time slice the requested percentage is recomputed as in step 1 and the process is repeated.

For a description of the SET command, see the *VM/SP Operator's Guide*.

If a percentage is not specified, a virtual machine with the favored execution option active is kept in the dispatch list except under the following conditions:

- Entering CP console function mode

- Loading a disabled PSW
- Loading an enabled PSW with no active I/O in process
- Logging on or off.

When the virtual machine becomes executable again, it is put back on the dispatch list in Q1. If dropped from Q1, the virtual machine is placed directly in the Q2 dispatch list. If the percentage option of the SET FAVORED command is specified, the deadline priority is calculated at queue drop time by:

$$\text{current time-of-day} + \frac{\text{length of allowed processor in-queue time slice}}{\text{favored percentage}}$$

For example, if the processor in-queue time slice is 1 second, and the specified percentage is 10 percent, then the value added to the current time-of-day is 10 seconds. The virtual machine should receive one processor time slice (1 second) once every 10 seconds.

Note, however, that these options can impact the response times of other virtual machines. To provide a virtual machine with both options, basic and percentage, both forms of the command for that virtual machine must be issued. The percentage form of the SET FAVORED command can be used to specify any number of logged-on virtual machines.

Although the SET FAVORED command prevents specifying more than 100% for a particular virtual machine, nothing is done to prevent allocating more than 100% to a number of virtual machines. Where more than 100% has been allocated, the favored virtual machines compete for the available resources on a pro-rata basis. An individual virtual machine's allocation is, roughly, proportional to the percentage allocated to it, divided by the total percentage allocated to all virtual machines. The effect of allocating more than 100% of the system on interactive (Q1) responses is unpredictable.

**Note:** The percentage of the processor time actually received by the favored user normally remains relatively close to the percentage specified in the command. However, it is not an absolute value, and varies depending on the total load on the system and the type of load on the system. If, for example, there are multiple virtual machines on the runlist that are compute bound (That is, are not queue dropped before the end of their in-queue time slice), then the favored user may not receive its requested percentage of the total processor time.

### *User Priority*

The VM/SP operator can assign specific priority values to different virtual machines. In so doing, the virtual machine with a higher priority is allocated a larger share of the system resources before a virtual machine with a lower priority. User priorities are set by the following class A command:

```
SET PRIORITY userid nn
```

where userid is the user's identification and nn is an integer value from 1 to 99. The value of nn affects the user's dispatching priority in relation to other users in the system. The priority value (nn) is one of the factors considered in the calculation of the deadline priority. The deadline priority is the basis on which all virtual machines in the system are ordered on both the eligible list and the dispatch list. The deadline priority calculation is based on the assumption that the average or normal (default) user priority is 64.

### *Reserved Page Frames*

VM/SP uses chained lists of available and pageable pages. Pages for users are assigned from the available list, which is replenished from the pageable list.

Pages that are temporarily locked in real storage are not available or pageable. The reserved page function gives a particular virtual machine an essentially "private" set of pages. The pages are not locked; they can be swapped, but only for the specified virtual machine. Paging proceeds using demand paging with a "reference bit" algorithm to select the best page for swapping. The number of reserved page frames for the virtual machine is specified as a maximum. The page selection algorithm selects an available page frame for a reserved user and marks that page frame "reserved" if the maximum specified for the user has not been reached. If an available reserved page frame is encountered for the reserved user selection, it is used whether or not the maximum has been reached.

The maximum number of reserved page frames is specified by a class A command of the following format:

```
SET RESERVE userid xxx
```

where xxx is the maximum number required. If the page selection algorithm cannot locate an available page for other users because they are all reserved, the algorithm forces the use of reserved pages. This function can be specified in only one virtual machine at any one time.

**Note:** xxx should never approach the total available pages, since CP overhead is substantially increased in this situation, and excessive paging activity is likely to occur in other virtual machines.

### *Virtual=Real*

For this option, the VM/SP nucleus must be reorganized to provide an area in real storage large enough to contain the entire virtual=real machine. In the virtual machine, each page from page 1 to the end is in its true real storage location; only its page zero is relocated. The virtual machine is still run in dynamic address translation mode, but since the virtual page address is the same as the real page address, no CCW translation is required. Since CCW translation is not performed, no check is made to ensure that I/O data transfer does not occur into page zero or any page beyond the end of the virtual=real machine's storage.

For information about generating a virtual=real system, see the *VM/SP Installation Guide*.

Figure 3 on page 33 is an example of a real storage layout with the virtual=real option. The V=R area is 128K and real storage is 512K.

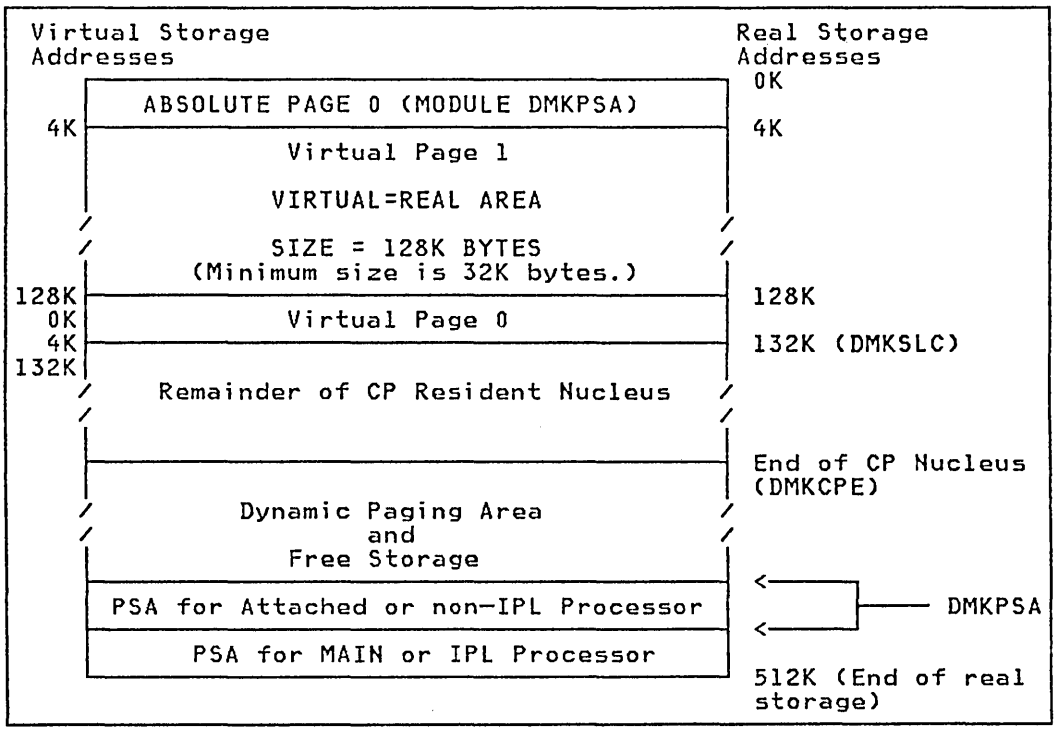



**Figure 3. Storage Layout in a Virtual=Real Machine**

There are several considerations for the virtual=real option that affect overall system operation:

1. The area of contiguous storage built for the virtual=real machine must be large enough to contain the entire addressing space of the largest virtual=real machine. The virtual=real storage size that a VM/SP system allows is defined during system generation when the option is selected.

2. The storage reserved for the virtual=real machine can only be used by a virtual machine with that option specified in the VM/SP directory. It is not available to other users for paging space, nor for VM/SP usage until released from virtual=real status by a system operator via the CP UNLOCK command. Once released, VM/SP must be loaded again before the virtual=real option can become active again.

3. The virtual machine with the virtual=real option operates in the preallocated storage area with normal CCW translation in effect until the CP SET NOTRANS ON command is issued. At that time, with several exceptions, all subsequent I/O operations are performed from the virtual CCWs in the virtual=real space without translation. The exceptions occur under any of the following conditions:

   - SIO tracing active
   - First CCW not in the V=R region
   - I/O operation is a sense command
   - I/O device is a dial-up terminal
   - I/O is for a nondedicated device
   - (spooled unit record console virtual CTCA or minidisks that are less than a full volume)
   - I/O device has an alternate path

- Pending device status

Any of the above conditions force CCW translation. Since minidisks are non-dedicated devices, they may be used by programs running in the V=R region even though CP SET NOTRANS ON is in effect.

4. If the virtual=real machine performs a virtual reset or IPL, then the normal CCW translation goes into effect until the CP SET NOTRANS ON command is again issued. This permits simulation of an IPL sequence by CP. Only the virtual=real virtual machine can issue the command. A message is issued if normal translation mode is entered.

5. A virtual=real machine is not allowed to IPL a named or shared system. It must IPL by device address.

6. When NOTRANS is in effect for a virtual=real machine, no meaningful SEEK data is collected by MONITOR operations.

7. If an installation defines a V=R area on a 3081 processor, the reliability and availability of the V=R machine can be improved if the V=R machine issues the TEST BLOCK instruction to validate storage in the V=R area. Note that the only two SCPs that issue TEST BLOCK are MVS/SP and VM/SP. The HSA area on a 3081 processor may reside in the middle of the V=R area; these two control programs mark the HSA segment as invalid and continue validating storage. Any other system control program, such as OS/VS, validates storage via the MVCL instruction. When OS/VS encounters the beginning of the HSA, it assumes that it has reached the end of storage. Therefore, such a control program running in the V=R area of VM/SP on a 3081 processor may not have access to the full V=R area.

8. If an installation intends to run in single processor mode on a 3081 processor, the system operator must issue a VARY OFF PROCESSOR nn VLOG command.

## *Affinity*

This option allows virtual machines that operate on attached processor or multiprocessor systems to select the processor of their choice for program execution. To select the affinity option, use the directory OPTION statement, or specify the AFFINITY operand on the class A, B, F, or G SET command. The directory OPTION statement is described in the *VM/SP Planning Guide and Reference*. The class A, B, and F SET commands are described in the *VM/SP Operator's Guide* and the class G SET command is described in the *VM/SP CP Command Reference for General Users*.

In application, the affinity setting of a virtual machine implies a preference of operation to either (or neither) processor. Affinity of operation for a virtual machine means that the program of that virtual machine will be executed on the selected or named processor. It does not imply that supervisory functions and the CP housekeeping functions associated with that virtual machine will be handled by the same processor.

In attached processor systems, all real I/O operations and associated interrupts are handled by the main processor. Virtual I/O initiated on the attached processor that is mapped to real devices must transfer control to the main processor for real I/O execution. Therefore, benefits may be realized in a virtual machine "mix" by

relegating those virtual machines that have a high I/O-to-compute ratio to the main processor, and those virtual machines that have a high compute-to-I/O ratio to the attached processor. Such decisions should be carefully weighed as every virtual machine is in contention with other virtual machines for resources of the system.

A system programmer can improve a virtual machine's performance on a multiprocessor where the path(s) to a user's primary minidisks exist from one processor only. In such cases, the system programmer could set the user's affinity to that processor.

A more important use of the affinity setting would be in applications where there are virtual machine program requirements for special hardware features that are available on one processor and not the other. Such features could be a performance enhancement such as virtual machine assist (described later in the text) or a special RPQ that is a requirement for a particular program's execution.

### *Multiple Shadow Table Support*

To reduce the number of purges when the virtual machine changes control register 1 (CR1) values, VM/SP maintains a queue of segment table origins (STO) and associated shadow tables for the virtual machine. Thus, each time an MVS or SVS system dispatches a new address space (changes CR1), VM/SP can dispatch the proper shadow table.

Multiple shadow table support adds one control block to VM/SP, the segment table origin control block (STOBLOK) pointed to by the ECBLOK. The STOBLOK, created by DMKVAT, contains all information pertaining to the shadow segment table, the shadow segment table itself and the virtual CR1 value. It also provides forward and backward queue pointers to the next STOBLOK on the queue. The first STOBLOK on the queue always contains the shadow STO to be loaded into CR1 when the virtual machine is dispatched in translation mode. The queue of STOBLOKs is maintained by DMKVAT in the following manner:

1. If a new CR1 value is loaded by the virtual machine, then the queue of STOBLOKs is searched for the virtual CR1 value.

2. If the proper STO is found, then the STOBLOK is ordered first on the queue.

3. If the proper STO is not found, then the maximum STO count is checked.

4. If the number of STOBLOKs equals the maximum STO count, then the last STOBLOK is stolen, the shadow tables are purged, and the STOBLOK is reinitialized and reused by being chained first on the queue with the new virtual CR1 value.

5. If the number of STOBLOKs is less than the maximum STO count, then free storage is obtained from VM/SP, and the STOBLOK is reinitialized and chained first on the queue.

Multiple shadow table support is controlled by the SET STMULTI command. The default shadow table count is 3 and the maximum is 6 per virtual machine.

### *Shadow Table Bypass*

Shadow table bypass is controlled by the SET STBYPASS command.

**Note:** If virtual machine assist is enabled on the system, the virtual machine must have the STFIRST directory option to be allowed to issue the SET STBYPASS nnM/nnnnK command.

**Shadow Table Bypass for the V=V User**

This technique is based on several characteristics of VS systems:

1. VS systems have a large area of addressing space starting with location zero where the virtual address is equal to the real address.

2. This addressing space is common to each segment table when multiple segment tables are used (MVS or SVS address space).

3. The VS system never pages within this fixed area.

Thus, an area starting with location zero can be established where the second-level address equals the third-level address or virtual-virtual=virtual-real (VV=VR). This allows a high-water mark, the highest VV=VR address, for a VS system to be established. Because the second-level address is the same as the third-level address, a reverse translation allows the shadow tables to be indirectly indexed. Then, whenever VM/SP steals a page from the VV=VR area, it invalidates the shadow page table entry and executes a real PTLB before redispatching the VS system's virtual machine.

In addition, whenever a shadow table is purged because a page frame is stolen from above the high-water mark or the virtual machine executed a PTLB or LCTL, the invalidation starts above the high-water mark, thus reducing purge and revalidation time.

**Shadow Table Bypass for the V=R User**

By the use of a V=R shadow table bypass technique, both the shadow tables and the overhead associated with maintaining them can be eliminated. This can be accomplished by VM/SP modifying the virtual operating system's page table to relocate virtual page zero to the highest real address within the V=R area. It is then possible to dispatch the virtual machine pointing to its own page and segment tables.

## *Queue Drop Elimination*

VM/SP attempts to optimize system throughput by monitoring the execution status of virtual machines. When a virtual machine becomes idle, VM/SP drops it from the active queue. The virtual machine's page and segment tables are scanned, and resident pages are invalidated and put on the flush list.

VM/SP determines that a virtual machine is idle when it voluntarily suspends execution (by loading a virtual PSW with the wait state bit on, for example), and no high-speed I/O operation is active. Normally, this is an adequate procedure.

However, in certain special cases, a virtual machine is determined to be idle and is queue dropped, but it becomes active again sooner than expected. If this cycle of queue dropping and reactivation is repeatedly executed, the overhead involved in invalidating and revalidating the virtual machine's pages may become large.

The SNA VTAM service machine is an example of this special case. The VTAM service machine operates by processing an IUCV message (or queue of messages),

and then suspending execution until the next message arrives. VM/SP queue drops the VTAM service machine when it suspends execution. When the next message arrives, all the VTAM service machine's pages must be revalidated. If the message rate is moderate to high, the overhead of repeated queue dropping exceeds the benefit.

The CP class A command "SET QDROP userid ON/OFF [USERS]" allows the installation to control this situation. If SET QDROP OFF is in effect for a virtual machine, the virtual machine remains active in the queue and its pages are not scanned or flushed. The page stealing mechanism is the only way the pages can be removed from storage. (Page stealing is invoked only if the flush list is empty.)

Specifying SET QDROP OFF for a service virtual machine may improve system performance and throughput when queue dropping would otherwise occur rapidly. But applying SET QDROP OFF indiscriminately may degrade system throughput by defeating the page flush mechanism and forcing page stealing to take place.

There can also be a relatively large overhead associated with a virtual machine being queue dropped during communications with a service machine for which the QDROP OFF specification is in effect. This can occur in small systems in which there is a high degree of virtual machine intercommunications. Specifying SET QDROP userid OFF USERS addresses this problem by providing for the temporary extension of the QDROP OFF status to any virtual machine communicating via VMCF or IUCV to the service virtual machine specified. The QDROP status for the "served" virtual machine remains in effect only while messages are outstanding between it and the service machine. Thus performance gains can be realized in systems with heavy usage of products such as IFS or PVM (invoked via the CMS PASSTHRU command). No additional performance gains will be realized in systems in which PVM is invoked via CP DIAL or with the SNA VTAM service machine, since the communication is with CP rather than another virtual machine.

The QUERY QDROP command (CP class A and E) may be used to list the userids for which SET QDROP OFF and the USERS parameter have been specified.

### *Virtual Machine Assist Feature*

The Virtual Machine Assist Feature is a processor hardware feature that improves the performance of VM/SP. Virtual storage operating systems, which run in problem state under the control of VM/SP, use many privileged instructions and SVCs that cause interrupts that VM/SP must handle. When the virtual machine assist feature is used, many of these interrupts are intercepted and handled by the processor. Consequently, VM/SP performance is improved.

The Virtual Machine Assist Feature intercepts and handles interruptions caused by SVCs (other than SVC 76), invalid page conditions, and several privileged instructions. An SVC 76 is never handled by the assist feature; it is always handled by CP. The processing of the following privileged instructions is handled by this feature:

| | |
|---|---|
| LRA | (load real address) |
| STCTL | (store control) |
| RRB | (reset reference bit) |
| ISK | (insert storage key) |
| SSK | (set storage key) |
| IPK | (insert PSW key) |
| STNSM | (store then AND system mask) |

| | |
|---|---|
| STOSM | (store then OR system mask) |
| SSM | (set system mask) |
| LPSW | (load PSW) |
| SPKA | (set PSW key from address) |

Although the assist feature was designed to improve the performance of VM/SP, virtual machines may see a performance improvement because more resources are available for virtual machine users. For a list of processors on which the Virtual Machine Assist Feature is available, see the *VM/SP Planning Guide and Reference.*

## *Using the Virtual Machine Assist Feature*

Whenever you IPL VM/SP on a processor with the virtual machine assist feature, the feature is available for all VM/SP virtual machines. However, the system operator's SET command can make the feature unavailable to VM/SP and, subsequently, available again for all users. If you do not know whether or not the virtual machine assist feature is available to VM/SP, use the class A and E QUERY command. For a complete description of the Class A and E QUERY and SET commands, see the *VM/SP Operator's Guide.*

If the virtual machine assist feature is available to VM/SP when you log on your virtual machine, it is also supported for your virtual machine unless you are running a second level VM/370 or VM/SP system in your virtual machine. If your VM/SP directory entry has the SVCOFF option, the SVC handling portion of the assist feature is not available when you log on. The class G SET command can disable the assist feature (or only disable SVC handling). It can also enable the assist feature, or if the assist feature is available, enable the SVC handling. You can use the class G QUERY SET command line to find whether you have full, partial, or none of the assist feature available. For a complete description of the Class G QUERY and SET commands, see the *VM/SP CP Command Reference for General Users.*

## *Restricted Use of the Virtual Machine Assist Feature*

Certain interrupts must be handled by VM/SP. Consequently, the assist feature is not available under certain circumstances. VM/SP automatically turns off the assist feature in a virtual machine that:

- Has set an instruction address stop
- Is tracing SVC and program interrupts

Since an address stop is recognized by an SVC interrupt, VM/SP must handle SVC interrupts while address stops are set. Whenever you issue the ADSTOP command, VM/SP automatically turns off the SVC handling portion of the assist feature for your virtual machine. The assist feature is turned on again after the instruction is encountered and the address stop removed. If you issue the QUERY SET command line while an address stop is in effect, the response indicates that the SVC handling portion of the assist feature is off.

Whenever a virtual machine issues a TRACE command with the SVC, PRIV, BRANCH, INSTRUCT, or ALL operands, the virtual assist feature is automatically turned off for that virtual machine. The assist feature is turned on again when the tracing is completed. If the QUERY SET command line is issued while SVCs or program interrupts are being traced, the response indicates the assist feature is off.

The Virtual Machine Assist Feature is not available to a second-level virtual machine, that is, a virtual machine that is running in a virtual machine.

## Extended Control-Program Support:VM/370 (ECPS)

Extended Control-Program Support:VM/370 (ECPS) extends, for specific privileged instructions, the hardware assistance that the virtual machine assist feature provides. ECPS also provides hardware assistance for frequently used VM/SP functions. The use of ECPS improves VM/SP performance beyond the performance gains that the virtual machine assist feature provides.

ECPS consists of three functions:

- CP assist
- Expanded virtual machine assist
- Virtual interval timer assist

CP assist provides hardware assistance for frequently used paths of specific CP functions.

Expanded virtual machine assist extends the hardware assistance that the virtual machine assist feature provides for the instructions LPSW, STNSM, STOSM, and SSM. In addition, expanded virtual machine assist provides hardware assistance for certain other privileged instructions.

Virtual interval timer assist provides hardware updating of the virtual interval timer at virtual address X'50'. Timer updating occurs only while the virtual machine is in control of the real processor. Virtual interval timer assist updates the virtual timer at the same frequency hardware updates the real timer, 300 times per second. Thus, virtual interval timer assist updates the virtual timer more frequently than CP updates it. Because the timer is updated more frequently, accounting routines may be able to provide accounting data that is more accurate.

ECPS does not support the same functions and instructions on all processors. Figure 4 lists the processors on which ECPS is available, and identifies, by processor, the functions and instructions ECPS supports.

| Functions and Instructions | 135-3, 138, 145-3<br>148, 4341 | 3031<br>3031AP | 4331 |
|---|---|---|---|
| **CP Assist** | X | X | |
| • Get free space (DMKFRE) | X | X | |
| • Release free space (DMKFRE) | X | X | |
| • Lock a page (DMKPTR) | X | X | |
| • Unlock a page (DMKPTR) | X | X | |
| • Test page status (DMKCCW) | X | X | |
| • Test page status and lock (DMKCCW) | X | X | |
| • Store ECPS identification | X | X | X |
| • SVC 8 (LINK) | X | X | X |
| • SVC 12 (RETURN) | X | X | X |
| • Scan for changed shared pages (DMKVMA) | X | X | X |
| • Locate virtual I/O control block (DMKSCN) | X | | X |
| • Invalidate page table (DMKVAT) | X | | X |
| • Invalid segment table (DMKVAT) | X | | X |
| • Untranslate CSW (DMKUNT) | X | | |
| • Free CCW storage (DMKUNT) | X | | |
| • Locate real I/O control block (DMKSCN) | X | | |
| • Common CCW command processing (DMKCCW) | X | | |
| • Decode first CCW (DMKCCW) | X | | |
| • Decode following CCW (DMKCCW) | X | | |
| • Main entry to dispatch (DMKDSP) | X | | |
| • Dispatch a block or a virtual machine (DMKDSP) | X | | |
| **Expanded Virtual Machine Assist** | | | |
| • LPSW | X | | |
| • STNSM | X | | |
| • STOSM | X | | |
| • SSM | X | | |
| • PTLB | X | | |
| • SIO | X | | |
| • SPT | | | |
| • SCKC | X | | |
| • STPT | | | |
| • TCH | X | X | |
| • DIAGNOSE | X | | X |
| **Virtual Interval Timer Assist** | X | X | X |

**Figure 4. Functions and Instructions that ECPS Supports**

### *Using the Extended Control-Program Support: VM/370*

Extended Control-Program Support: VM/370 (ECPS) is controlled at two levels: the VM/SP system and the virtual machine.

At the VM/SP system level, ECPS is automatically enabled when the system is loaded. The class A command:

```
set cpassist off
```

disables both CP assist and expanded virtual machine assist. The class A command:

```
set sassist off
```

disables only the expanded virtual machine assist part of ECPS as well as the virtual machine assist. CP assist is the only part of ECPS that is truly independent.

At the virtual machine level, whenever ECPS is enabled on the system, both expanded virtual machine assist and virtual interval timer assist are automatically enabled when you log on. If you issue the class G command:

```
set assist off
```

both assists as well as the existing virtual machine assist are disabled. If you issue:

```
set assist notmr
```

only the virtual interval timer assist is disabled. If CP assist is disabled for the system, the class A command:

```
set sassist on
```

enables the virtual machine assist. You can then enable virtual machine assist and virtual interval timer assist for your virtual machine by issuing the class G command:

```
set assist on tmr
```

#### Restricted Use of ECPS

The restrictions on the use of ECPS are the same as those described for the virtual machine assist feature with one addition. Whenever a virtual machine traces external interrupts, the virtual interval timer assist is automatically disabled. When external interrupt tracing is completed, virtual interval timer assist is reenabled.

## Channel Usage

### *The Virtual Block Multiplexer Channel Option*

Virtual machine SIO operations are simulated by CP in three ways:

- byte-multiplexer
- selector
- block multiplexer channel mode.

Virtual byte-multiplexer mode is reserved for I/O operations that apply to devices allocated to channel zero.

In virtual selector channel operations, CP reflects a busy condition (condition code 2) to the virtual machine's operating system if the system attempts a second SIO to the same device, or another device on the same channel, before the first SIO is completed.

Block multiplexer channel mode is a CP simulation of real block multiplexer operation; it allows the virtual machine's operating system to overlap SIO requests to multiple devices connected to the same channel. The selection of block multiplexer mode of operation may increase the virtual machine's throughput, particularly for those systems or programs that are designed to use the block multiplexer channels.

**Note:** CP simulation of block multiplexer processing does not reflect channel available interruptions (CAIs) to the user's virtual machine.

Selecting the channel mode of operation for the virtual machine can be accomplished by either a DIRECTORY OPTION operand or by use of the CP DEFINE command.

## *Multisystem Communications*

The IBM 3088 Multisystem Channel Communication Unit (MCU) is an input/output device that interconnects as many as eight systems using block multiplexer channels. The 3088 Model 1 interconnects up to four systems, while the 3088 Model 2 interconnects up to eight systems.

The 3088 supports the PREPARE channel command which can be used in certain situations to prevent attention interrupts on the side issuing the PREPARE command. See "Channel Command Words" later in this discussion for a description of 3088 channel command words.

The 3088 is compatible with existing channel-to-channel usage. In addition, 3088 support extends existing CTCA addressing and scheduling by:

- Allowing multiple unit addresses per control unit
- Implementing block multiplexer channel scheduling for both real and virtual CTCAs and 3088.

### System Programmer Considerations

At system generation time, the system programmer must code parameters in the RDEVICE macro and the RCTLUNIT macro to define the 3088 to the control program. See the *VM/SP Installation Guide* for the format of these macros.

***RDEVICE MACRO***: When you code the RDEVICE macro, specify the address and device type. For example, to define a maximum of 32 sequential unit addresses at A00, code the RDEVICE macro as follows:

```
RDEVICE ADDRESS=(A00,32),DEVTYPE=3088
```

***RCTLUNIT MACRO***: When you code the RCTLUNIT macro, you must indicate the address and the control unit type. In addition, since the 3088 supports a maximum of 32 or 64 devices, you must also specify the number of sequential unit addresses using the FEATURE=xxx-DEVICE operand. For example, if you want to generate 32 devices at channel address A00, code the RCTLUNIT macro as follows:

```
RCTLUNIT ADDRESS=A00,CUTYPE=3088,FEATURE=32-DEVICE
```

***SPECIAL DIRECTORY CONTROL STATEMENT***: The 3088 is a valid device for the SPECIAL directory control statement. For example, to specify a 3088 at virtual address A00, code the SPECIAL directory control statement as follows:

```
SPEcial A00 3088
```

### Virtual 3088 Support

Use the class G DEFINE command to define a virtual 3088 device. The class G user can define a virtual 3088 device with or without a real equivalent. The system simulates all functions of the real 3088, except for the online testing functions, for each virtual 3088 that you define. You must define each virtual 3088 unit address with a single DEFINE command. Defining each virtual unit address is different from the dedicated 3088 support where you can define multiple unit addresses using a single RDEVICE macro. Refer to *VM/SP Operating Systems in a Virtual Machine* for examples of virtual machine usage of channel-to-channel devices.

### Command Usage and 3088 Support

Support for the 3088 recognizes the 3088 as a valid device. Figure 5 outlines commands affected by 3088 support. See the *VM/SP Operator's Guide* and the *VM/SP CP Command Reference for General Users* for the format and complete discussion of these commands.

| Command | Class | 3088 Support |
|---|---|---|
| DEFINE | G | The 3088 is a valid device type on this command. The control unit address for a CTCA and a 3088 need not end in zero. Once you define the control unit, you may define other virtual devices for the same CTCA or 3088. |
| ATTACH<br>COUPLE<br>DETACH<br>QUERY | B<br>G<br>G, B<br>B | The response to these commands is the same for channel-to-channel adapters (CTCAs) and 3088s. |

**Figure 5. CP commands and 3088 Support**

### Channel Command Words

In addition to the channel commands supported in System/360 and System/370 modes, the 3088 supports the following two channel commands:

- PREPARE -- the PREPARE channel command is used to receive a channel program without causing an attention interrupt to the side issuing the command.
- SENSE ID -- the SENSE ID channel command transfers model and control unit identification to the system issuing the command.

3088 support offers online testing facilities, and messages and MNOTES as diagnostic aids when using the support. See *VM/SP System Message and Codes* for the complete text of the messages.

***ONLINE TESTING***: The last address in the group of 32 or 64 addresses for each interface attached to the 3088 is available as a dedicated diagnostic unit address. The diagnostic unit address provides a communication path between diagnostic programs and the 3088 microprocessor for online testing. For example, a system attached to the 3088 may use the diagnostic unit address to read the 3088 logout and error information.

***MESSAGES AND MNOTES TO SUPPORT 3088 DEVICES***: The system issues a message or MNOTE in the following situations:

- If you attempt to define a 3088 for a unit address that has previously been defined
- If the virtual channel-to-channel device specified in the COUPLE command is busy on the receiving userid's virtual machine
- If you attempt to couple a 3088 to a channel-to-channel adapter
- If you specify a model on the RDEVICE macro.

## *Alternate Path Support*

Through the use of the Two-Channel Switch and Two-Channel Switch Additional Features, alternate path support for DASD or tape provides for up to four channels on one control unit to be attached to VM/SP (up to 2 channels per control unit in multiprocessing configurations). In addition, one device may be attached to two logical control units, providing support for the String Switch feature. This allows the control program up to eight paths to a given device when the maximum number of alternate channels and alternate control units are specified.

When an I/O request is received for a device that has alternate paths defined, and the primary path is unavailable, VM/370 searchs for the first available path beginning with the first alternate path. Successive alternate paths are examined if required until an available path is found. In the case where no available path to the device exists, alternate path I/O scheduling is implemented to queue the request on multiple busy/scheduled paths, and the first path to become available is the path the I/O request is started on.

The VM/370 I/O Scheduler determines that a path is available by analyzing the busy and scheduled software indicators in the RDEVBLOK, RCUBLOK, and RCHBLOK as well as the chains of pending I/O requests that are queued from the RCUBLOK and RCHBLOK. This processing is performed prior to issuing the SIO.

The search for an available path begins with the RDEVBLOK. If the RDEVBLOK is marked busy or scheduled, the I/O request is queued on the RDEVBLOK. No alternate path scheduling is performed at the device level. If the RDEVBLOK is not busy or scheduled, the IOBLOK for this I/O request is promoted upward to the RCUBLOK. If the RCUBLOK is marked busy, the IOBLOK is queued on the RCUBLOK and a search is made for an alternate control unit

path. If the RCUBLOK is marked scheduled and the present request will not release the control unit (example: TAPE FSF and TAPE BSF), the IOBLOK is queued off the RCUBLOK and a search is made for an alternate control unit path. If the RCUBLOK is marked scheduled and the present request will release the control unit, the search continues for a channel path. If the RCUBLOK is not marked scheduled or busy but there are other I/O requests queued on the RCUBLOK, the check is again made to see if the present request will release the control unit. If the present request will not release the control unit, the request is queued and a search is made for an alternate control unit path. Otherwise, the search continues for a channel path.

The RCUBLOK "busy" and "scheduled" indicators are only turned on for shared control units. The busy and scheduled indicators are turned on in the RCUBLOK for tape and 2314 DASD control units. The non-shared DASD RCUBLOKS never have the busy and scheduled indicators in the "on" status. For this reason, alternate control unit path selection rarely takes place for non-shared control units. The one exception occurs when the channel path through the first control unit appears busy (because a real channel busy condition was encountered). If an alternate path exists through a second control unit, the control blocks associated with the second control unit path are examined.

Finding an available channel path is the final step prior to issuing the SIO. If the RCHBLOK is marked busy, a search is made for an alternate channel path. If the RCHBLOK has other requests queued on the RCHBLOK, a search for an alternate channel path is made. VM/370 never marks a byte multiplexor RCHBLOK busy. The only time a block multiplexor is marked busy is after a condition code 2 has been encountered. The I/O load on block multiplexor channels must be sufficient to cause channel busy conditions before path selection on an alternate channel can take place.

## MVS/System Extensions Support

The MVS/System Extensions support in VM/SP allows an MVS system running in a virtual machine to exploit the enhancements available in the MVS/System Extensions Program Product (Program No. 5740-XE1) if the System/370 Extended Facility or System/370 Extended Feature is present on the hardware.

Included in the MVS/System Extensions Program Product enhancement is the use of:

1. The System/370 Extended Facility for the 303x and the 308x processors, or
2. The System/370 Extended Feature for the System/370 Model 158 and 168 processors, or
3. ECPS:MVS for the 4341.

**Note:** An RPQ (MK3272) is available for the 158-3 processor that allows the coexistence of virtual machine assist and System/370 Extended Facility (S370E) and VM/370 Extended Feature. Thus, an MVS/SE virtual machine can run under VM/SP with virtual machine assist active on a 158-3 processor. ECPS:MVS and ECPS:VM/370 are mutually exclusive in the 4341 Model Group 1 and 4341 Model Group 2. The control storage expansion feature of the Model Group 2 allows coexistence of ECPS:MVS and ECPS:VM/370.

The System/370 Extended Facility and System/370 Extended Feature, and ECPS:MVS are enabled by the MVS/System Extensions support as defined by the directory OPTION statement or via the CP SET command. For details, refer to the *VM/SP Operator's Guide*, and the *VM/SP CP Command Reference For General Users*, respectively.

MVS/System Extensions support includes:

- Low address protection facility[1]
- Common segment facility[1]
- Special MVS instruction operation facilities

### *Low Address Protection Facility*

The low address protection facility provides protection against improper storing by instructions using logical storage addresses in the range 0-511. This facility prevents inadvertent program destruction of those storage locations that the processor uses to fetch new PSWs during interruption processing. Low address protection does not apply to the storing of status by the processor (for example, old PSWs, logout data), nor does it apply to any channel stores (for example, CSW or LCL).

Bit 3 of control register 0 is defined as the low address protection bit, and is used to control whether or not stores using logical addresses in the range 0 to 511 are permitted. When this bit is zero in real control register zero, stores are permitted; when this bit is one, stores are not permitted. When an instruction attempts a store using an address in the range 0 to 511 and low address protection applies, the contents of the storage area addressed by the instruction are not modified. The execution of the current instruction is terminated or suppressed, and a protection exception occurs.

### *Common Segment Facility*

The common segment facility allows addressing segments to be classified as private or common. If bit 30 of the segment table entry for a given segment is 1, the segment is a common segment; otherwise it is private. A private segment table entry and the page table it designates may be used only in association with the segment table origin (STO) that designates the segment table in which the segment table entry resides. A common segment table entry and the page table it designates may continue to be used for translating addresses even though a different STO is specified by changing control register 1.

### *Special MVS Instruction Operation Handling Facilities*

Special operations and instructions in the MVS/System Extensions Program Product that enhance MVS operations are handled by System/370 Extended Facility or System/370 Extended Feature, and are described in the IBM publication *System/370 Extended Facility*, GA22-7022. Invalidate Page Table Entry (IPTE) and Test Protection (TPROT) instructions described in this publication are simulated in VM/SP.

---

1 ECPS:MVS is identical to the Extended Facility, except that the Low Address Protection Facility and the Common Segment Facility are not included.

### *Enabling MVS/System Extensions Support*

Using the class A SET S370E ON command, the system operator enables the MVS/System Extensions support for all virtual machines. Using the class G SET 370E ON command (or 370E option on the directory OPTION control statement), the general user enables the support for a particular virtual machine.

### *Single Processor Mode*

When an OS/VS2 MVS Attached Processor (AP) system or an OS/VS2 MVS tightly-coupled Multiprocessing (MP) system runs on a multiprocessor under VM/SP, without using single processor mode, MVS runs in uniprocessor mode. That is, MVS programs do not execute simultaneously on both processors. Therefore, MVS does not attain the level of throughput it could attain were it running in multiprocessor mode.

To improve the throughput of an OS/VS2 MVS AP system or OS/VS2 MVS MP system, run MVS in the V=R machine and use single processor mode. Running in this mode, MVS has exclusive use of one processor while VM/SP and the V=R machine (running MVS) use the other processor. In other words, MVS runs on two processors instead of one. This improves MVS's throughput.

The throughput of an OS/VS2 MVS AP or OS/VS2 MVS MP system running under VM/SP and using single processor mode is higher than the throughput would be were single processor mode not used. However, single processor mode may reduce the throughput of VM/SP and virtual machines not using the V=R area.

Single processor mode cannot improve the throughput of a VM/SP attached processor or multiprocessor system. A VM/SP AP or MP system initialized (by IPL) in the V=R machine with single processor mode on runs in uniprocessor mode.

Two commands provide operator control of single processor mode. SPMODE, a class A command, turns single processor mode on or off. QUERY, a class A or G command, indicates whether single processor mode is on or off.

For detailed instructions on how to turn single processor mode on or off, see *VM/SP Operating Systems in a Virtual Machine.*

### *Dynamic System Control Programming (SCP) Transition to or from Native Mode*

There are times when an installation benefits from switching an SCP to or from native mode. For example, when it is important to obtain the best possible performance from an SCP, switch it to native mode. When there is a need to do different kinds of work simultaneously, switch the SCP from native mode to the VM/SP environment.

Installations have always had the capability to switch an SCP to or from native mode, but to do so has been time consuming. Switching an SCP to native mode meant quiescing the SCP and VM/SP and then initial program loading the SCP. To return the SCP to the VM/SP environment meant quiescing the SCP and then initial program loading VM/SP and the SCP.

Dynamic SCP transition to or from native mode enables an operator to dynamically switch an SCP to or from native mode. Switching to native mode, there is no long-

er a need to quiesce or reinitialize (via IPL) the SCP. The SCP continues to run and can do productive work. Switching back to the VM/SP environment, there is no longer a need to quiesce the SCP or IPL VM/SP or the SCP.

Before switching an SCP to or from native mode, an operator must prepare VM/SP and the SCP for the switch: for example, all users except the VM/SP operator and the operator on the V=R machine must logoff VM/SP. Detailed instructions on preparing the systems and on switching to or from native mode are in *VM/SP Operating Systems in a Virtual Machine.* The following discussion highlights the switching process and defines precautions that must be observed.

To switch an SCP to native mode, it must be running in the V=R machine. The VM/SP operator then prepares VM/SP and the SCP for the switch. To complete the switch, the operator issues the QVM command (quiesce VM).

After the switch to native mode is completed, there are two areas of real storage that must not be altered. Addresses 0-7 contain the restart PSW (program status word) used to make the transition back to the VM/SP environment. Storage above the upper limit of the V=R area contains the VM/SP nucleus. Altering either area may make it impossible to return to the VM/SP environment.

To return the SCP to the VM/SP environment, an operator uses the System/370 restart facility. After stopping the processor, the operator stores the value X'FF' into the real storage address located eight bytes prior to the address pointed to by the restart PSW. To complete the switching process, the operator restarts the processor. **Caution:** This process does not work unless the SCP was switched to native mode via the QVM command.

The performance of an SCP switched to native mode depends on the size of the V=R area. The SCP's performance will be identical to the performance it would attain were it initialized (via IPL) directly on a hardware configuration identical to the V=R machine's configuration with a real storage size equal to the storage size of the V=R area. In other words, the larger the V=R area, the better the SCP performs.

You can can switch to or from native mode using the procedures just described for:

- OS/VS1 running without VM VS1 Handshaking
- OS/VS2 SVS
- OS/VS2 MVS

# Performance Observation and Analysis

Three commands, INDICATE, QUERY SRM, and MONITOR, provide a way to dynamically measure system performance.

***Indicate:*** Provides the system analyst and general user with a method to observe the load conditions on the system while it is running.

***QUERY SRM:*** Provides the system operator with expanded observation facilities for analyzing internal activity counters and parameters.

***Monitor:*** Provides the system analyst and the system operator with a data collection tool designed for sampling and recording a wide range of data. The collection of data is divided into functional classes. The different data collection functions can be performed separately or concurrently. Keywords in the MONITOR command enable the collection of data and identify the various data collection classes. Other keywords control the recording of collected data on tape for later examination and reduction.

## Load Indicators

The INDICATE command allows the system operator to check the system for persistently heavy loads. The operator can, therefore, judge when it is best to apply additional scheduling controls (if appropriate) or call a system analyst to perform an analysis of the condition by using the INDICATE, QUERY SRM, and MONITOR commands.

The system analyst has a set of operands in the INDICATE command that displays the basic uses of and contentions for major system resources (possible bottleneck conditions) and identifies the userids and characteristics of the active users and the resources that they use.

Virtual machine users can use the INDICATE command to observe the basic smoothed conditions of contention and utilization of the primary resources of processor and storage. The INDICATE command allows them to base their use of the system on an intelligent guess of what the service is likely to be. Over a period of time, virtual machine users relate certain conditions of service to certain utilization and contention figures, and know what kind of responses to expect when they start their terminal session.

## The INDICATE Command

The INDICATE command allows general users and the system analyst to display at their consoles at any time, the usage of and contention for major system resources.

General users can display usage of and contention for the major system resources of processor and storage. They can also display the total amount of resources used during the terminal session and the number of I/O requests. If they use the INDICATE command before and after the execution of a program, users can determine the execution characteristics of that program in terms of resource usage.

The system analyst can identify active users, the queues they are using, their I/O activity, their paging activity, and many other user characteristics and usage data.

The system analyst can use the data on system resource usage and contention to monitor the performance of the system. The analyst can thus be aware of heavy load conditions or low performance situations that may require the use of more sophisticated data collection, reduction, and analysis techniques for resolution.

The VM/SP Scheduler maintains exponentially smoothed values for data provided by the LOAD option. Specifically, at intervals (in seconds) depending on the processor model, the scheduler calculates the total activities for variables such as CP and storage usage for the most recent interval, and factors them into a smoothed wait value in the following way:

$$\text{New smoothed wait value} = \frac{(3 * \text{old smoothed wait value}) + \text{current interval wait}}{4}$$

Thus, only 1/4 of the most recent interval wait is factored into the new smoothed wait which makes it predominantly the old smoothed wait value.

The remaining INDICATE components are sampled prior to a user being dropped from a queue. Because of the frequency of this event, the remaining components are subject to a heavier smoothing than the wait time. A general expression for the smoothing follows:

```
nsv = ((rate - int) (osv) / rate) + civ
```

*where:*

nsv = new smoothing value

osv = old smoothing value

civ = current interval value (results found during the current interval (int))

int = current interval (time period being tested)

rate = either history interval (hrate) of 8 minutes, or data interval (drate) of 75 seconds

Other operands of the command allow users to obtain other performance information that enables them to understand the reasons for the observed conditions. For the format of the class G INDICATE command, see the *VM/SP CP Command Reference for General Users*. For the format of the class E INDICATE command, see the *VM/SP Operator's Guide*.

## The INDICATE FAVORED Command

The section "Preferred Virtual Machine Options" in this publication contains detailed information on favored execution. For information on the setting of favored execution options, refer to the *VM/SP Operator's Guide*.

## Managing Page Migration

In order to keep 12% of the preferred paging area available, CP migrates inactive pages from preferred to nonpreferred paging areas. The preferred paging area includes a fixed-head area and a moveable-head area. The fixed-head paging area is paging space on a drum and/or space under the fixed heads of a DASD volume

that has the fixed head feature installed. The moveable-head paging area is paging space on a DASD volume that is accessed by a moveable arm. Normally, CP dynamically invokes page migration, based on calculated load levels, once every ten minutes.

Inactive pages in the fixed-head preferred paging area are migrated every time CP invokes migration. For pages in the moveable head preferred paging area, the installation can decide at what point inactive pages are selected for migration. The system programmer can use the SET SRM MHFULL command to set moveable head page migration limits.

If a percentage for MHFULL has been specified, CP migrates pages from moveable head preferred paging areas only when that percentage is reached and the time interval has elapsed, rather than whenever fixed head areas are full. Thus, migration from moveable head preferred paging areas and fixed head preferred paging areas can take place separately.

In addition, the system operator can use the MIGRATE command to immediately invoke page and swap table migration. Page migration can also be invoked only for a specific virtual machine.

The format of the MIGRATE command is described in the *VM/SP Operator's Guide.*

## Querying and Setting the System Resource Management Variables

The QUERY SRM and SET SRM commands allow the system analyst to query and/or change internal system activity counters or parameters. Formats for the QUERY SRM and SET SRM commands are contained in the *VM/SP Operator's Guide.*

The system analyst can use the Class E QUERY SRM command to display the following information:

- Current number of pageable pages
- Size of the dispatching time slice
- Setting of the maximum working set estimate
- Maximum drum page allocation limit
- Current page migration counters
- Unused segment elapsed time as criteria for page migration
- Current PCI flag setting mode for 2305 page requests
- Maximum page bias value
- Current interactive shift bias value
- Moveable head page migration limit

The class E SET SRM command allows the system analyst to set some of the system variables that can affect the values displayed by the QUERY SRM command.

### *Querying and Setting the Paging Variable*

The paging variable is used in the working set size algorithm. The current paging load is constantly compared with the paging variable. Adjustments are then made in the working set size estimates, based on how well the actual load compares with the paging load variable.

The QUERY PAGING command displays the paging variable used in the working set size estimate control algorithm. Information on the paging rate per second is available as a response to the INDICATE LOAD command.

The SET PAGING command is used to change the paging variable used in the working set size estimate.

Information about the formats of the QUERY PAGING and SET PAGING commands is contained in the *VM/SP Operator's Guide*.

## The MONITOR Command

VM/SP Monitor collects data in two ways:

1. By handling interruptions caused by executing MONITOR CALL (MC) instructions.

2. By using timer interruptions to give control periodically to sampling routines.

MONITOR CALL instructions with appropriate classes and codes are presently embedded in strategic places throughout the main body of VM/SP code (CP). When a MONITOR CALL instruction executes, a program interruption occurs if the particular class of MONITOR CALL is enabled. The classes of MONITOR CALL that are enabled are determined by the mask in control register 8. For the format and function of the MONITOR CALL instruction, refer to the *System/370 Principles of Operation*. The format of control register 8 is as follows:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| xxxx | xxxx | xxxx | xxxx | 0123 | 4567 | 89AB | CDEF |

*where:*

**x** indicates unassigned bits.

**0-F (hexadecimal)** indicates the bit associated with each class of the MONITOR CALL.

When a MONITOR CALL interruption occurs, the CP program interruption handler (DMKPRG) transfers control to the VM/SP Monitor interruption handler (DMKMON) where data collection takes place.

Sixteen classes of separately enabled MONITOR CALL instructions are possible, but only eight are implemented in the VM/SP Monitor.

Monitor output consists of event data and sampled data. Event data is obtained via MONITOR CALL instructions placed within the VM/SP code. Sampled data is collected following timer interruptions. All data is recorded as though it were obtained through a MONITOR CALL instruction. This simplifies the identification of the records.

The following table indicates the type of collection mechanism for each Monitor class:

---

2 There is no class name for monitor class 3, but it is reserved.

| Monitor Class | Class Name | Collection Mechanism |
|---|---|---|
| 0 | PERFORM | Timer requests |
| 1 | RESPONSE | MC instructions |
| 2 | SCHEDULE | MC instructions |
| 3[2] | -- | -- |
| 4 | USER | Timer requests |
| 5 | INSTSIM | MC instructions |
| 6 | DASTAP | Timer requests |
| 7 | SEEKS | MC instructions |
| 8 | SYSPROF | Collected via class 2 |

Another function, separate from the VM/SP Monitor, is also handled by the MONITOR command. The MONITOR command can stop and start CP internal trace table data collection, which is *not* initiated by MONITOR CALLs.

**Note:** The VM/SP Monitor record format and the contents of the record are shown in "Appendix B. Monitor Tape Format and Content."

The class A and E MONITOR command:

- Stops and starts CP internal trace table data collection.
- Displays the status of the internal trace table and each implemented class of VM/SP Monitor data collection. Displays the specifications for automatic monitoring defined by the SYSMON macro in DMKSYS. In addition, it displays those specifications for automatic monitoring that are overridden by Monitor commands. It also displays whether the tape or spool file is the recording medium.
- Starts and stops VM/SP data collection using tape or spool file. It also closes the spool file, if desired.
- Specifies VM/SP monitor classes of data collection enabled, number of buffers used, and time of data collection. It also specifies other options which override the specifications for automatic monitoring on the SYSMON macro contained in DMKSYS.
- Specifies the interval to be used for timer driven data collection.
- Specifies direct access devices that are to be included or excluded from a list of devices. The list defines direct access devices for which CP is to collect data for the SEEKs class.

See the *VM/SP Operator's Guide* for the format and details of the MONITOR command.

### *Implemented Classes*

The following MONITOR CALL classes correlate with the corresponding classes in control register 8. Refer to the *System/370 Principles of Operation* for details of the MC instruction and the bits in control register 8.

| Monitor Class | Keyword | Data Collection Function |
|---|---|---|
| 0 | PERFORM | Samples system resource usage data by accessing system counters of interest to system performance analysts. |
| 1 | RESPONSE | Collects data on terminal I/O. Simplifies analyses of command usage, user, and system response times. It can relate user activity to system performance. This class is invalid and no data can be collected for it unless the system programmer changes the LOCAL COPY file and reassembles DMKMCC. |
| 2 | SCHEDULE | Collects data about scheduler queue manipulation, monitors flow of work through the system, and indicates the resource allocation strategies of the scheduler. |
| 3 | ------- | Reserved. |
| 3 | USER | Periodically scans the chain of VMBLOKs in the system, and extracts user resource utilization and status data. |
| 5 | INSTSIM | Records every virtual machine privileged instruction handled by the control program (CP) standard simulation routines (DMKPRV, DMKPRW). Because simulation of privileged instructions is a major source of overhead, this data may lead to methods of improving performance.<br><br>The fast path simulation routines (DMKFPS) result in significantly less control program overhead than the standard paths. Therefore, privileged instructions simulated by DMKFPS are not recorded.<br><br>If the VMA feature is active, the number of privileged instructions that are handled by the control program is reduced for those virtual machines that are running with the feature activated. |

| Monitor Class | Keyword | Data Collection Function |
|---|---|---|
| 6 | DASTAP | Periodically samples device I/O activity counts (SIOs), for tape and DASD devices only. DASTAP samples only those tapes and DASD devices that are online when the MONITOR START command is issued.<br><br>It is possible that the number of DASD and tape devices defined in DMKRIO may exceed 291 (the maximum number of MONITOR DASTAP records that fit in a MONITOR buffer). The following algorithm determines which devices are monitored:<br><br>1. If the total number of DASD and tape devices that are online is less than or equal to 291, all online DASD and tape devices are monitored.<br><br>2. If the total number of online DASD devices is less than or equal to 291, all online DASD devices are monitored.<br><br>3. Otherwise, the first 291 online DASD devices are monitored. |
| 7 | SEEKS | Collects data for every I/O request to DASD. Reveals channel, control unit, or device contention and arm movement interference problems.<br><br>**Note:** When NOTRANS is in effect for a virtual=real machine, no meaningful data is collected.<br><br>No data is collected for TIO or HIO operations. For SIO operations, data is collected when the request for the I/O operation is initially handled and again when the request is satisfied.<br><br>This means that a single SIO request could result in two MONITOR CALLs. For example, if the request gets queued because the device is already busy, then a MONITOR CALL would be issued as the request is queued. Later, when the device becomes free and is restarted, a second MONITOR CALL is issued.<br><br>In general, the data collected is the same except that in the first case nonzero counts are associated with queued requests.<br><br>If the request for I/O is satisfied when it is initially handled without being queued, only one MONITOR CALL results. In both this case and the second of the two data collections mentioned above, the count of I/O requests queued for the device is zero. |

| Monitor Class | Keyword | Data Collection Function |
|---|---|---|
| 8 | SYSPROF | Collects data complementary to the DASTAP and SCHEDULE classes in order to provide a more detailed "profile" of system performance through a closer examination of DASD utilization. |

## *VM/SP Monitor Response to Unusual Tape Conditions*

### Suspension

When I/O to the tape is requested, the device may still be busy from the previous request. If this occurs, two data pages are full and data collection must be temporarily suspended. Control register 8 is saved and then set to zero to disable MONITOR CALL program interruptions and timer data collection. A running count is kept of the number of times suspension occurs. The current Monitor event is disregarded. When the current tape I/O operation ends, the next full data page is scheduled for output. MONITOR CALL interruptions are reenabled (control register 8 is restored), a record containing the time of suspension, the time of resumption, and the suspension count is recorded and data collection continues. The suspension count is reset to zero when the MONITOR STOP TAPE is issued. If a MONITOR command is issued at the time monitor is suspended, a message is displayed to the invoker of the command stating,

```
SEEK, STOP, OR CLOSE CMD IS IN PROGRESS, RETRY
```

### Unrecoverable Tape Error

When an unrecoverable error occurs, DMKMON receives control and attempts to write two tape marks, rewind, and unload the tape. The use of the tape is discontinued and data collection stops. The operator is informed of the action taken. Whether or not the write-tape-marks, rewind, and unload are successful, the tape drive is released.

### End-of-Tape Condition

When an end-of-tape condition occurs, DMKMON receives control. A tape mark is written on the tape and it is rewound and unloaded. The VM/SP Monitor is stopped and the operator is informed of the action taken.

## *VM/SP Monitor Considerations*

The system programmer may want to set the TRACE(1) bit to a 1 in the LOCAL COPY file and reassemble DMKMCC to allow RESPONSE data (MONITOR class 1) to be collected. See the information about security exposure in "MONITOR ENABLE Restrictions" in the MONITOR command description.

### Initial Program Load

MONITOR START CPTRACE is active after real system IPL (manual or automatic). The VM/SP Monitor tape data collection is off after IPL. If automatic performance monitoring is specified in the SYSMON macro and IPL occurs within the range of the TIME operand of the SYSMON macro, VM/SP monitor data collection to a spool file is started.

### System Shutdown

If the VM/SP Monitor data collection to a spool file is taking place, a system shutdown causes closing of the file and termination of monitoring. If data collection is to tape, a system shutdown implies a MONITOR STOP TAPE command. Normal command processing for the MONITOR STOP TAPE function is performed by the system.

### System Failure

If the VM/SP system fails and data collection to a spool file is active, the spool file is closed and preserved, except for the last buffer. If the VM/SP system fails and data collection is active on tape, an attempt is made to write two tape marks, rewind, and unload the tape. If the tape drive fails to rewind and unload, be sure to write a tape mark before rewinding and unloading the tape. VM/SP Monitor data collection is terminated by the system failure.

### I/O Devices

If VM/SP monitor data collection is active using tape, a supported tape drive must be dedicated to the system for the duration of the monitoring. For accounting purposes, all I/O is charged to the system.

## *VM/SP Monitor Data Volume and Overhead*

Use of the VM/SP Monitor usually requires that three pages be locked in storage for the entire time the VM/SP Monitor is active; however, only two pages are required if the single buffer option is used with only the PERFORM class of data collection enabled. This reduces by three the number of page frames available for paging. This significantly affects the performance of the rest of the system when there is a limited number of page frames available for paging.

PERFORM This class of data collection is activated once every 60 seconds (or as defined by the MONITOR INTERVAL command), and records system counters relevant to performance statistics. It is, therefore, a very low overhead data collection option.

RESPONSE This class collects terminal interaction data and, because of the human factor, has a very low rate of occurrence relative to processor speeds. Consequently, this class causes negligible overhead and produces a low volume of data.

SCHEDULE This class records the queue manipulation activity of the scheduler and generates a record every time a user is added to the eligible list, added to queue1 or queue2, or removed from queue. The recording overhead is very low.

USER This class of data collection is active once every 60 seconds (or as defined by the MONITOR INTERVAL command). Data is extracted from each user's VMBLOK, including the system VMBLOK. The overhead incurred is comparable with that of the statistical data of the PERFORM class; however, it increases with the number of users logged onto the system.

INSTSIM This class of data collection can give rise to large volumes of data because of the frequency of privileged instructions in some virtual machines. This may incur significant overhead. It should be activated for short periods of time and preferably, though not necessarily, when other classes of data collection are inactive. If the Virtual Machine Assist feature is active for the virtual machine, the data volume and, consequently, the CP overhead may be reduced.

DASTAP This class of data collection samples device activity counts once every 60 seconds (or as defined by the MONITOR INTERVAL command) and is a very low source of overhead, similar to the PERFORM and USER classes.

SEEKS This class of data collection can give rise to large volumes of data because every start I/O request to DASD is recorded via a MONITOR CALL.

SYSPROF This class of data collection is complementary to the SCHEDULE and DASTAP classes and results in a small amount of additional overhead. It obtains more refined data on DASD resource usage.

### Performance for Time-Shared Multibatch Virtual Machines

First you must determine how many similar users can be run concurrently on a given configuration before the throughput of individual users becomes unacceptable.

### Monitoring Recommendations

Every installation should use the automatic monitoring facilities to simplify and automate the collection of performance data. A virtual machine should also be set up to analyze and report the collected data. The VM/370 Performance/Monitor Analysis Program (VMAP) does such a task. For more information about the capabilities of this program and for details about ordering it, see the publication *Virtual Machine Facility/370 Performance/Monitor Analysis Program*. This program or user-written analysis programs should be run on a daily basis to analyze the collected data. Data reduction should preferably be run at off-peak hours to minimize the effect on the performance of the system that is doing data reduction. Initially, the data collected with MONITOR default options should be analyzed to establish a familiarity with the load environment and performance profile of each virtual machine system and its effect on CP.

Once a performance profile is established for each system and associated virtual machines, the analyst should be able to detect points of contention between processor(s) storage, I/O, and paging subsystems.

Normally the spool file monitoring options should be used. However, if large volumes of trace data are to be collected, then monitoring to tape should be used. Tape is also useful if benchmarking is frequently done and all of the new monitor trace and sampled data must be archived for possible future use. The default mode of operation of the Performance/Monitor Analysis Program is to keep the condensed ACUM files and not the raw data.

If SEEKs data is needed, a sampling technique is suggested. A simple implementation might be to use a CMS EXEC procedure to enable SEEKs for ten seconds every ten minutes. This would produce SEEKs data while limiting the volume of data collected. An alternative is to create a list of devices for which data for the SEEKs class is to be collected. CP collects data for only those devices in the list. To create the list, use the INCLUDE or EXCLUDE options of the MONITOR command's SEEK operand. If data is collected for only a few devices, consider collecting data for longer periods of time.

## *Load Environments of VM/SP*

Two distinct uses of VM/SP can be readily identified and, consequently some differences in criteria for acceptable performance may occur. The system may be required to time share multiple batch-type virtual machines with interactive machines performing minor support roles; or, the system may be primarily required to provide good interactive time-sharing services in the foreground, with a batch background absorbing spare resources of real storage and processor.

After determining the minimum acceptable performance, perform external observations of turnaround time on benchmarks and specify a point beyond which the addition of more users would be unacceptable. However, when that point is reached, more sophisticated internal measurement is required to determine the scarcest resource and how the bottleneck can be relieved by additional hardware.

Several possible conditions can be identified resulting from different bottlenecks. They are:

- Real storage levels of multiprogramming are low compared with the number of contending users. Hence, each user is dispatched so infrequently that running time or response time may become intolerable.

- Storage may be adequate to contain the working sets of contending users, but the processor is being shared among so many users that each is receiving inadequate attention for good throughput.

- Real storage space may be adequate for the processor, and a high speed drum is used for paging; however, some virtual storage pages of some users have spilled onto slower paging devices because the drum is full. With low levels of multiprogramming, user page wait can become a significant portion of system wait time. Consequently, processor utilization falls and throughout deteriorates.

- Storage, processor, and paging resources are adequate, yet several users are heavily I/O-bound on the same disk, control unit, or channel. In these circumstances, real storage may be fully committed because the correct level of multiprogramming is selected, yet device contention is forcing high I/O wait times and unacceptable processor utilization.

Estimates of typical working set sizes are needed to determine how well an application may run in a multiprogramming environment on a given virtual storage system. A measure of the application's processor requirements may be required for similar reasons. Measurements may be required on the type and density of privileged instructions a certain programming system may execute, because, in the virtual machine environment, privileged instruction execution may be a major source of overhead. If the virtual machine environment is used for programming development, where the improvement in programmer productivity outweighs the disadvantages of extra overheads, the above points may not be too critical. However, if throughput and turnaround time are important, then the converse is true, and the points need close evaluation before allocating resources to a virtual machine operation.

High levels of multiprogramming and overcommitment of real storage space lead to high paging rates. High paging rates can indicate a healthy condition; but be concerned about page stealing and get evidence that this rate is maintained at an acceptable level. A system with a high rate of page stealing is probably thrashing.

*Performance -- Mixed Mode Foreground/Background Systems with Emphasis on Good Interactive Response*

Most of the conditions for good performance, established for the time-shared batch systems, apply equally well to mixed mode systems. However, two major factors make any determination more difficult to make. First, get evidence to show that, in all circumstances, priority is given to maintaining good interactive response, and that nontrivial tasks truly take place in the background. Second, background tasks, no matter how large, inefficient, or demanding should not be allowed to dominate the overall use of the time-sharing system. In other words, in mixed mode operation, get evidence that users with poor characteristics are discriminated against for the sake of maintaining a healthy system for the remaining users.

A number of other conditions are more obvious and straightforward. You need to measure response and determine at what point it becomes unacceptable and why. Studies of time-sharing systems have shown that a user's rate of working is closely

correlated with the system response. When the system responds quickly, the user is alert, ready for the next interaction, and thought processes are uninterrupted. When the system response is poor, the user becomes sluggish.

For interactive environments, a need exists to analyze command usage. Average execution time of the truly interactive commands can provide data for validation of the Queue 1 execution time.

## Trace Table Recording Facility

The Trace Table Recording Facility provides Field Engineering and system programmers with problem determination capability. The facility, using the CPTRAP command, creates a READER spool file of selected trace table entries, CP data entries, and virtual machine data entries in the order they happen. A CMS data reduction program, TRAPRED, is also provided to examine the data collected. Output can be either a spooled print file or an interactive terminal display.

### Recording CP Trace Table Entries in the CPTRAP File

The current CP internal trace table is a 'wrap' table that continuously overlays previously stored information. The trace table recording facility continues to store trace table entries in the trace table and, in addition, to the spool under command control. The trace records are selectable by trace type with the CPTRAP command.

Current trace record types are:

| | |
|---|---|
| 01 - External interrupt | 0E - Test I/O |
| 02 - SVC interrupt | 0F - Halt device |
| 03 - Program interrupt | 10 - Unstack IOBLOK or TRQBLOK |
| 04 - Machine check interrupt | 11 - NCP BTU |
| 05 - I/O interrupt | 12 - Spinning on lock |
| 06 - Free storage | 13 - SIGP issued |
| 07 - Return storage | 14 - Clear Channel issued |
| 08 - Enter scheduler | 15 - IUCV Communication |
| 09 - Queue drop | 16 - SNA Console Comm. Services |
| 0A - Run user | 17 - MSSF Support |
| 0B - Start I/O | 18 - Start I/O Fast Release |
| 0C - Unstack I/O interrupt | 19 - Simulated I/O interrupt |
| 0D - Virtual CSW store | 1B - Clear I/O |

The trace table recording facility allows selection of input by trace table type (typenum) and further selection based on a field in the trace table entry. The three allowed fields are VMBLOK address, device address (DEVADDR), and code

(CODE). Selectivity by typenum is allowed up to X'3F', inclusive, to allow for future trace types. Further selectivity using VMBLOK, DEVADDR, and CODE, is only defined for the CP trace types. Selected CP trace table entries are moved to the spool file without change to their format or length.

**Passing Virtual Machine Data to the CPTRAP File**

This interface allows the virtual machine to enter data into the CPTRAP spool file. The virtual machine interface is inserted into a problem area by a FE or system programmer using the STORE CP command. When enabled, via the CPTRAP ALLOWID subcommand, the data area is paged in by the CPTRAP facility, but must be an address within the virtual machine's storage. The data length can be variable up to a maximum of 264 bytes. Longer requests are honored only for the first 264 bytes with no indication that data length has been truncated.

The interface is via a class 10 monitor call instruction. The user should use monitor code 0 with this instruction. Therefore, any virtual machine running user written programs that use a class 10 monitor call should not use the CPTRAP virtual machine interface facility because of unpredictable results.

A monitor call instruction can be executed in virtual supervisor or virtual problem state, BC mode or EC mode, and in multi-level environments. Multi-level is defined as VM/SP running a guest virtual machine (GVM) of a VM/SP system. There is no restriction on the number of interfaces that can be active at once, or the number of virtual machines that can use them. Within VM/SP, the trace facility maintains an ordered file of the virtual interface information and selected VM/SP control program (CP) information.

The first byte of a virtual interface record is assigned a type X'3E', for virtual machine records. The second byte is reserved. The third and fourth bytes contain the user-defined code to individualize this record. The fifth and sixth bytes contain the length. The length is the full record including the type indicator, reserved bytes, two code bytes, the two length bytes, and the data itself. The seventh and eighth bytes are reserved. The data starts on the ninth byte of the record to supply doubleword alignment. The total record length is a multiple of 16 bytes to keep incore trace record alignment. This facilitates reading the nontransferred records from a dump. A time stamp is put on every 4K spool block.

Register 1 must contain a pointer to a PARM field that contains a pointer to the data, the length of the data, and the individualizing code. The data must be 264 bytes or less, and reside in the virtual machine. If the length is greater than 264 bytes, only the first 264 are taken.

The individualizing code exists so that interface data may be identified at data reduction time. It is the user's responsibility to make it a unique indication of where the interface was inserted. To use the virtual machine interface, each virtual machine must be enabled via the ALLOWid operand of the class C CPTRAP command. For a complete description of the CPTRAP command and its operands, see the *VM/SP Operator's Guide*.

The interface is:

MC 0,10 with Register 1 set as follows:

R1 is a pointer to an 8-byte parameter list whose format is:

| LLLL | CODE | DATA ADDR |
|---|---|---|

where:

LLLL is a two byte field containing the length of the data field

CODE is a two byte field containing the individualizing code

DATA ADDR is a four byte pointer to the data field to be included on the spool file.

**Note:** There are no boundary requirements for the 8-byte parameter list.

An example of acceptable logic: where '3' is the individualizing code, and DATAFLD is the address of the data.

```
         BAL   R1,AROUND
         *                      PARM DEFINITION
         DC    AL2(L'DATAFLD)   2 BYTES OF LENGTH FIELD
         DC    AL2(3)           2 BYTES OF CODE...THIS IS CODE 3
         DC    AL4(DATAFLD)     4 BYTES OF DATA POINTER
AROUND   DS    0H
         MC    0,10             VIRTUAL MACHINE INTERFACE
```

### Passing CP Data to the CPTRAP File

To insert the CP interface into the problem area, insert the code into the source, reassemble the particular source module, and regenerate the system. You can also use the STCP command to insert the CP interface into a problem area. There are no restrictions on the number of interfaces that may be active at the same time.

The interface is via a BALR R14,R15. Register 15 must be loaded with the address of PSA(TRAPOK); this is the address of logic within module DMKPSA. This logic determines if CPTRAP is active. If active, a branch is taken to the CPTRAP logic via PSA(TRAPCP); if not, a return is immediately taken to the caller.

Register 1 must contain a pointer to the parameter list that contains the length of the data, an individualizing code, and a pointer to the data. The data must be 264 bytes or less, and reside in real storage. If the length is greater than 264 bytes, only the first 264 are taken with no indication that the data length was truncated.

The individualizing code exists so that CP interface data may be identified at data reduction time. It is the user's responsibility to make it a unique indication of where the zap was inserted.

Eight bytes of data are added by CP as a header for the CP interface record. The first byte is set to X'3F' to indicate CP interface data, the second is reserved. The third and fourth bytes contain the individualizing code, the fifth and sixth bytes contain the total length (data length plus 8), the seventh and eighth bytes are reserved and are immediately followed by data.

Care must be taken where the trap is inserted. If a condition code is set which has not yet been interrogated, the inserted interface logic must assure that the condition code is saved or unchanged. The CP interface logic within CPTRAP does preserve the condition code setting.

The interface is:

R1 a pointer to an 8-byte parameter list whose format is:

| LLLL | CODE | DATA ADDR |
|---|---|---|

where:

LLLL is a two byte field containing the length of the data field.

CODE is a two byte field containing the individualizing code.

DATA ADDR is a four byte pointer to the data field to be included on the spool file.

R15 must contain the address of TRAPOK in the PSA

R14 contains the return address.

**Note:** There are no boundary requirements for the 8-byte parameter list.

The following is an example of acceptable logic where '3' is the individualizing code, and DATAFLD is the address of the data.

```
         USING PSA,R0
         BAL   R1,AROUND
         *                          DATA STRUCTURE
         DC    AL2(L'DATAFLD)       2 BYTES OF LENGTH FIELD
         DC    AL2(3)               2 BYTES OF CODE...THIS IS CODE 3
         DC    AL4(DATAFLD)         4 BYTES OF DATA POINTER
AROUND   DS    0H
         LA    R15,TRAPOK
                                    * CP INTERFACE
         BALR  R14,R15              * METHOD
          .
          .
         PSA
```

### CPTRAP READER File

The spool file created by the trace table recording facility has a filename of CPTRAP and a filetype of FILE. It is made up of noncontiguous 4K spool records. The records are all data; there are no CCWs within them. The file has the SFBDUMP bit on, and the SFBMISC+1 byte is C'P'. The file may be accessed with DIAGNOSE X'14'.

### CMS Data Reduction Program

A reduction program, which runs on CMS, is included with the facility to allow the data collected by CPTRAP to be useable. Input is the spool file created by the CPTRAP command. Output can be either a spooled print file or an interactive terminal display. The program allows:

- Selectivity of output
- Positioning within the file
- Displaying on the terminal in a forward or backward direction

- Printing
- Stopping

The data is unblocked into logical records and printed in hexadecimal. Each logical record starts on a new line. The interface data, the only data that can be longer than one print line, is indented to improve its readability. The program, TRAPRED, is a CP module shipped with the CMS modules, and exists on the CMS S disk with an S2 filemode. TRAPRED writes CP error messages to the terminal, and writes line mode data to the terminal, and to the printer. Although TRAPRED issues CP messages, the program runs in the CMS user area.

| TRAPRED | filenum |
|---|---|

In CMS mode enter:

```
TRAPRED filenum
```

where:

TRAPRED
is the name of the CPTRAP data reduction program.

filenum
is the number of the READER file that is the output of CPTRAP processing. The filename and filetype are 'CPTRAP FILE'.

The spool file specified must belong to the user invoking the program, and must be of a class which can be read and must not be held.

After you enter 'TRAPRED filenum', the CPTRAP READER file is accessed and interactive processing may begin. The CMS immediate commands HT and HX work with this program.

There are three categories of interactive subcommands:

```
All         [ON ]

All         [OFF]

typenum     [Vmblok  nnnnnn]
            [DEVaddr nnnn  ]
            [COde    nnnn  ]
            [OFF           ]
```

These subcommands control which trace table types are to be viewed on the terminal or spooled to the printer. The specification of X'3F' or X'3E' for typenum causes selectivity of CP or virtual machine interface records respectively. Remember that the records on the file were selectively chosen at CPTRAP invocation, so this may be looked at as a second level of selectivity.

All [ON]
: All selectivity is desired at this time (data reduction time). All records on the READER file are processed. This is the default setting at invocation time if selectivity is not modified. OFF turns off all selectivity definitions. ALL or ALL ON turns on all typenum general selectivity.

| All [OFF] | No selectivity, no records will be processed. |
|---|---|
| typenum | Vmblok nnnnnnnn<br>DEVaddr nnnn<br>COde nnnn<br>OFF |

These have the same meaning as during CPTRAP selectivity definition. In addition, typenums X'3F' and X'3E' may be specified for CP interface data and virtual machine interface data, respectively.

When the first typenum is entered after invoking the program or "ALL[ON]", all other typenums are reset to off.

**File Processor Subcommands**

When the TRAPRED program is invoked, the message 'ENTER SELECTIVITY OPTION(S) OR SUBCOMMAND' is issued.

| | |
|---|---|
| TOP | |
| BOTtom | |
| Up | nnnnnnnn |
| DOwn | nnnnnnnn |
| Type | nnnnnnnn |
| TYPEBack | nnnnnnnn |
| Printer | nnnnnnnn |

`TOP/BOTtom` Use these commands for positioning within the CPTRAP READER file. TOP positions to the first record of the file; BOTtom at the last record.

`Up/Down nnnnnnnn`

1 Use these commands to move data around within the READER file. The amount of movement is controlled by the number following the command. The number indicates how many logical records are to be skipped; counting is selective. DOWN is toward the end of the file, UP is toward the start of the file. The nnnnnnnn may be any decimal value between 1 and 99999999. Wrapping of the file is not allowed, and the subcommand ends prematurely if either the end of file or start of file is encountered.

`Type nnnnnnnn`

1 Type indicates that selective output is to be displayed on the terminal. nnnnnnnn is the number of selective lines desired. If unspecified, one line, the current line, is displayed. nnnnnnnn may be any decimal value between 1 and 99999999. The command ends prematurely if the end of file is encountered.

For CP and virtual machine interface records (types X'3F' and X'3E' respectively), the indicated number of logical records is displayed. Each logical record may be more than one line.

`TYPEBack nnnnnnnn`

1 This is the same as TYPE except instead of proceeding in a forward direction, records are selectively displayed in a backward direction within the file, that is, toward the top of the file. The command ends prematurely if the start of file is encountered.

For CP and virtual machine interface records (types X'3F' and X'3E' respectively), the indicated number of logical records is displayed. Each logical record may be more than one line.

`Printer nnnnnnnn`

1 Use this command to spool selected records to the printer. nnnnnnnn is the number of records to print. 1 is the default, but you may specify any decimal value up to 99999999. The command ends prematurely if the end of file is encountered. It proceeds toward the bottom of the file.

### End Program Commands

`QUIT`
The subcommand to end data reduction. The accessed reader file is kept.

`STOP`
Used to end data reduction. The file is purged unless the reader is held.

### Lost Data

A data lost message is issued when the system creates output faster than it can be transferred to the spool. When this happens, the output file also indicates that data has been lost. The amount of data lost can be 4K of CP trace table and/or CP/virtual data records. The possibility of a data lost situation is:

1. Directly proportional to the rate of transfer of trace table data to spool
2. Directly proportional to the frequency and size of interface data
3. Inversely proportional to speed of the spool DASD. This is a potential problem with the faster CPUs and/or with heavy use of the interface support. A reduced selection of trace types and CP or virtual interface data is the solution.

### Checkpointing

Closed CPTRAP reader files are checkpointed just like any other spool files.

### AP and MP Support

AP and MP processors are supported by the trace table recording facility. When the facility is actively processing the CP trace table, the AP/MP lock becomes active. The control method is a word within the module which is tested with a TS (TEST and SET) instruction. This word also holds the CPU identification of the latest CPU through the CPTRAP logic.

### Running with Microcode Assist Active

The trace table recording facility requires the dispatcher assists in ECPS:VM/370 to be deactivated to support the monitor call interface for virtual machines. These assists exist in DMKDSP. Deactivation is required only when the facility is active, and is accomplished automatically by tests just in front of the assist instructions.

### Logoff Considerations

The trace table recording facility is stopped if the user who invoked CPTRAP logs off.

# Accounting Records

The accounting data gathered by VM/SP can help in analysis of overall system operation. Also, accounting data can be used to bill VM/SP users for time and other system resources they use.

There are three types of accounting records: the virtual machine user records, records for dedicated devices as well as T-disk space assigned to virtual machine users, and accounting records generated as a result of user initiated DIAGNOSE X'4C' instruction. A CMS batch virtual machine creates an accounting record with the userid and account number of the user who sent his job to the batch machine. Accounting records are prepared as 80-character card images and spooled to disk.

When the user wishes, the data can be sent to the punch for punched output, or spooled to the virtual machine's reader for additional processing. By using the SYSACNT macro, the user can do this when a specified number of records are accumulated. By invoking the ACNT CLOSE command, the user does it immediately.

## Accounting Records for Virtual Machine Resource Usage

The information stored in the accounting record in card image form when a user ends his terminal session (or when the ACNT command is invoked) is as follows (columns 1-28 contain character data; all other data is in hexadecimal form, except as noted):

| *Column* | *Contents* |
|---|---|
| 1- 8 | Userid |
| 9-16 | Account number |
| 17-28 | Date and Time of Accounting (mmddyyhhmmss) |
| 29-32 | Number of seconds connected to VM/SP System |
| 33-36 | Milliseconds of processor time used, including time for VM/SP supervisor functions |
| 37-40 | Milliseconds of virtual processor time used |
| 41-44 | Number of page reads |
| 45-48 | Number of page writes |
| 49-52 | Number of virtual machine SIO instructions for nonspooled I/O |
| 53-56 | Number of spool cards to virtual punch |
| 57-60 | Number of spool lines to virtual printer (This includes one line for each carriage control command) |
| 61-64 | Total number of spool records from virtual reader (This is not the number of records read, rather it is the total number of records in the spool file (SFBRECNO) when the file is open for processing.) |
| 65-78 | Reserved |
| 79-80 | Accounting record identification code (01) |

## Accounting Records for Dedicated Devices and Temporary Disk Space

Accounting records are recorded and spooled to disk when a previously dedicated device and temporary disk space is released by a user via DETACH, LOGOFF, or releasing from DIAL (dedicated device only). A dedicated device is any device assigned to a virtual machine for that machine's exclusive use. These include devices dedicated by the ATTACH command, those being assigned at logon by directory entries, or by a user establishing a connection (via DIAL) with a system

that has virtual 2702 or 2703 lines. The information on the accounting record in card image form is as follows (columns 1-28 contain character data; all other data is in hexadecimal form, except as noted):

| Column | Contents |
|---|---|
| 1- 8 | Userid |
| 9-16 | Account number |
| 17-28 | Date and Time of Accounting (mmddyyhhmmss) |
| 29-32 | Number of seconds connected to VM/SP system |
| 33 | Device class |
| 34 | Device type |
| 35 | Model (if any) |
| 36 | Feature (if any) |
| 37-38 | Number of cylinders of temporary disk space used (if any) or number of blocks used (columns 37-40) for fixed-block devices. This information appears only in a code 03 accounting record. |
| 39-78 | Unused (columns 41-78 unused for fixed-block devices) |
| 79-80 | Accounting record identification code (02, 03) |

The device class, device type, model, and feature codes in columns 33-36 are shown in Figure Figure 69 on page 521.

## Accounting Records for LOGON, AUTOLOG, and LINK Journaling

When LOGON, AUTOLOG, and LINK journaling is on, VM/SP may write type 04, type 05, type 06, or type 07 records to the accounting data set. These records are written under the following circumstances:

- Type 04 records are written when VM/SP detects that a user has issued enough LOGON or AUTOLOG commands with an invalid password to reach or exceed an installation defined threshold value.
- Type 05 records are written when VM/SP detects that a user has successfully issued a LINK command to a protected minidisk not owned by that user.
- Type 06 records are written when VM/SP detects that a user has issued enough LINK commands with an invalid password to reach or exceed an installation defined threshold value.
- Type 07 records are written when a user logs off a device controlled by the VCNA. The records indicate the user's share of the VCNA resource used.

These records have the following formats:

**Type 04**

| Column | Contents |
|---|---|
| 1- 8 | USERID specified on the command |
| 9-16 | Reserved for IBM use |
| 17-28 | Date and time of accounting (mmddyyhhmmss) |
| 29-32 | Terminal address |
| 33-40 | Invalid password |
| 41-48 | USERID that issued the AUTOLOG command |
| 49-51 | Reserved for IBM use |
| 52-53 | Current invalid password count |

| | |
|---|---|
| 54-55 | Accounting record limit (JPSLOGAR) |
| 56-78 | Reserved for IBM use |
| 79-80 | Accounting card identification code (04) |

**Type 05**

| Column | Contents |
|---|---|
| 1- 8 | USERID that issued the command |
| 9-16 | Account number |
| 17-28 | Date and time of accounting (mmddyyhhmmss) |
| 29-32 | Terminal address |
| 33-40 | Reserved for IBM use |
| 41-48 | USERID of user that owns the minidisk |
| 49-51 | Minidisk address for which the LINK command was issued |
| 52-78 | Reserved for IBM use |
| 79-80 | Accounting card identification code (05) |

**Type 06**

| Column | Contents |
|---|---|
| 1- 8 | USERID that issued command |
| 9-16 | Account number |
| 17-28 | Date and time of accounting (mmddyyhhmmss) |
| 29-32 | Terminal address |
| 33-40 | Invalid password |
| 41-48 | USERID of user that owns the minidisk |
| 49-51 | Minidisk address for which the LINK command was issued |
| 52-53 | Invalid password count |
| 54-55 | Invalid password limit (JPSLNKAR) |
| 56-78 | Reserved for IBM use |
| 79-80 | Accounting card identification code (06) |

**Type 07**

| Column | Contents |
|---|---|
| 1- 8 | USERID or terminal identification |
| 9-16 | Accounting number or 0000 |
| 17-78 | VM/VCNA accounting data |
| 79-80 | Accounting card identification (07) |

## Accounting Records Created by the User

A virtual machine user can initiate the creation of an accounting record that contains up to 70 bytes of information of his own choosing. To do this, he issues a DIAGNOSE code X'4C' instruction with the following operands:

- The address of a data area in virtual storage containing the information, in the actual format, that he wishes to have recorded in columns 9 through 78 of the card image record.
- A hexadecimal function code of X'10'
- The length of the data area in bytes

The information on the accounting record is as follows:

| Column | Contents |
|---|---|
| 1- 8 | Userid |
| 9-78 | User formatted data |
| 79-80 | Accounting record identification code (C0) |

For information on using DIAGNOSE code X'4C' see "DIAGNOSE Instruction in a Virtual Machine" in this section.

For SNA users, VM/VTAM Communications Services (VCNA) uses the VM/SP user accounting record. See the *VCNA Installation and Terminal Use Guide* for the format of this record.

## User Accounting Options

You may insert your own accounting procedures in the accounting routines. See the "CP Conventions" section for information on CP coding conventions and loadlist requirements. Operator responsibilities in such cases should be defined by the installation making the additions. When designing such accounting procedures, you should understand that:

1. The accounting routines are designed to be expanded. The entry point provided in the accounting module for installation use is called DMKACON. If you want to perform additional accounting functions, you should modify the following copy files:

   ACCTON (account on) -- for action at logon time. This is provided as a null file. It can be expanded to provide additional functions at logon time. The ACCTON routine can request the system to force the user off by returning a nonzero value in SAVER2. However, if the operator is automatically logged on during system initialization, the nonzero return code has no effect.

   **Note:** The ACCTON COPY file distributed with VM/SP contains the basic logic required to enhance system security based on the 3277 Operator Identification Card Reader feature. Additional checking may be added to examine or validate the data read from the identification card.

   ACCTOFF (account off) -- for action at logoff time. This section contains the code that fills in the account card fields. It does not reset any internal data. This file exists in both DMKACO and DMKCKP (checkpoint). If the ACCTOFF copy file is changed, both modules should be reassembled.

2. In addition to CP accounting, your installation can use the accounting routines to supply virtual machine operating system accounting records. This provides a means of job accounting and operating system resource usage accounting.

3. If you specify, in the SYSACNT system generation macro, that your spooled accounting records are to be sent to the reader of a virtual machine, you can process the accounting data directly with your own accounting routines.

# Generating Saved Systems

By taking advantage of the SAVESYS command, system resources are not committed to perform an IPL each time a system is loaded. Instead, the saved system is located and page tables are initialized according to its system name table entry. The saved system is not automatically loaded at IPL time; however, its pages are brought into storage on demand as the virtual machine operating system executes.

In addition to saving time by avoiding an IPL, a saved system can share segments of reenterable code, thus making more efficient use of real storage. This technique is especially valuable when using CMS. However, a shared segment cannot be initialized in the virtual=real machine, by an IPL.

To generate a saved system:

- Add the appropriate NAMESYS or NAMENCP macro and operands to the DMKSNT file
- Assemble this new version of DMKSNT
- Create and load a new control program nucleus
- IPL the new CP system
- Load the system to be saved and then issue the SAVESYS command.

When allocating DASD space for named systems, provide an extra page for information purposes; do not overlay this area with subsequent named systems. See the *VM/SP Planning Guide and Reference* and the *VM/SP Installation Guide* for further information on generating and saving saved systems.

## The NAMESYS Macro for Saved Systems

The NAMESYS macro is assembled by the installation system programmer and is used to describe the location of the saved system. Shared segments may be specified, but they must consist of reenterable code.

When making additions, changes, or deletions to the system name table, the DMKSNT module must be reassembled. The GENERATE EXEC procedure has the facility to reassemble only the DMKSNT module. See the description of the GENERATE EXEC procedure in the *VM/SP Installation Guide.*

A DMKSNT ASSEMBLE sample supplied with the system contains workable CMS segments. Either edit or update this module to include the NAMESYS macros describing your installation's named systems. Note that this module may contain a PUNCH SPB card, which is used by the loader to force this module to a 4K boundary when the CP system is built (a 12-2-9 multipunch must be specified in column 1 of an SPB).

### *Coding the NAMESYS Macro*

The NAMESYS macro describes the name and location of the saved system or discontiguous saved segment. Shared segments may be specified, but they must consist of reenterable code, with no alteration of its storage space permitted. See the *VM/SP Planning Guide and Reference* for the format of the NAMESYS macro.

### *Example of a DMKSNT Entry*

A DMKSNT entry to create a named CMS system could be coded as follows:

```
DMKSNTBL CSECT
FSTNAME  NAMESYS  SYSNAME=CMS,SYSVOL=VMSRES,SYSSTRT=(001,1),    X
                  SYSPGNM=(0-4,14-33,400-511),SYSPGCT=137,      X
                  SYSHRSG=(25,26,27,28,29,30,31),SYSSIZE=256K,  X
                  VSYSADR=190,SYSCYL=98,VSYSRES=VMSRES          X
         END
```

In the above example, VMSRES is a count-key-data volume (IBM 3330 Disk Storage).

A similar example in which the device is a fixed-block volume follows:

```
DMKSNTBL CSECT
FSTNAME  NAMESYS  SYSNAME=CMS,SYSVOL=VMSRES,SYSSTRT=(2),         X
                  SYSPGNM=(0-4,14-33,400-511),SYSPGCT=137,      X
                  SYSHRSG=(25,26,27,28,29,30,31),SYSSIZE=256K,  X
                  VSYSADR=190,SYSBLOK=46912,VSYSRES=VMSRES      X
         END
```

If the segment resides on one volume and the virtual 190 minidisk resides on a different volume, it is possible for one of the devices to be fixed-block and the other to be count-key-data. In that case, the parameters pertaining to each device are selected based on the type of that particular device.

## Using the SAVESYS Command

The system to be saved must first be loaded by device address in the traditional manner. Before its page-format image can be saved, the system to be saved must have its execution stopped. The point at which the operating system is stopped should be determined by the installation system programmer. The SAVESYS command must then be issued; its format is:

| SAVESYS | systemname |
|---|---|

*where:*

`systemname` corresponds to the identification of the saved system. This is identical to the SYSNAME entry in the NAMESYS macro.

The user must have a CP privilege class of E to issue the SAVESYS command. Next, he should IPL the saved system. The virtual machine will attempt to resume execution and immediately encounter a page fault. The required page is brought into storage and execution continues. As execution continues, subsequent page faults bring the required pages into storage.

A system should be saved as soon after IPL as possible. All pages to be saved must be resident at the time the SAVESYS command is issued. Also, before issuing the SAVESYS command, be sure that the system is stopped.

CMS was designed to run under CP and it was also designed so that it could easily be saved by CP. See "Saving the CMS System" in "Part 2. Conversational Monitor System (CMS)" of this publication.

**Note:** The system being saved should not exceed X'79C000' bytes. Unpredictable results may occur if you save a larger system.

## Shared Segments

If one or more segments of a saved system are designated as being "shared," a single copy of these segments in real storage can be used by any virtual machine that loads the saved system by name. (In attached processor or multiprocessor mode, there are two sets of pages, page tables, and swap tables maintained for each shared segment.) A shared segment must be reenterable and the segment number must be included in the SYSHRSG operand of the NAMESYS macro for the saved system.

If, for example, you code the SYSHRSG= operand of the NAMESYS macro for the system to be saved as

```
SYSHRSG=(29,30,31)
```

then segments 29, 30, and 31 of the system are to be shared. When CMS is saved, via the SAVESYS command, the pages in segments 29, 30, and 31 are set up so that any user loading the saved system by name shares the same set of these pages in real storage. This results in a saving of both real and external page storage. Also, the more virtual machines using the shared segment, the more likely it is that these pages will be frequently referenced and, thereby, kept in real storage. As a result, the number of page faults and the corresponding time and resources expended in page swapping is reduced.

### *Special Considerations for Shared Segments*

When a saved system containing one or more shared segments is again saved, a problem can occur if the previous system has been loaded by name and is still in use. If users of the "old" system continue to reference pages that have already been brought into paging storage, no problems will occur. However, if after the new system has been saved, users of the old system reference pages that had not previously been referenced, they receive the *new* version of the referenced page.

Any users who IPL the newly saved system share only the new copy of the shared segment.

Also, the entire segment is saved by the SAVESYS command, not just that portion occupied by the program (for example, CMS), so that unwanted data may also be contained in the segment.

The use of shared segments is not allowed in a virtual=real machine.

The maximum number of shared segments that may be defined is 78.

## Discontiguous Saved Segments

With discontiguous saved segment support, you can attach and detach segments of storage to and from your virtual machine. These segments contain reenterable code that can be shared by many users. Thus, programs that are required sometimes, but not all the time, can be shared and only loaded when they are needed.

Segments that are to be shared in this manner must be loaded at an address beyond the normal end of your virtual machine and then must be saved. The procedure for loading and saving discontiguous segments is similar to the procedure that already exists for loading and saving systems. Also, discontiguous saved segments can be attached to your virtual machine in nonshared mode for testing and debugging. In summary, a discontiguous saved segment is a segment that:

- Has a name associated with it
- Contains only reenterable code
- Was previously loaded and saved
- Can be shared by multiple virtual machines
- Can be loaded by a particular virtual machine in nonshared mode for testing and debugging

**Note:** A discontiguous saved segment must not be attached by a virtual machine executing in the virtual=real area.

An example of a discontiguous saved segment is the segment of CMS that supports DOS program development and testing under CMS. This segment is reenterable and is named CMSDOS. The VM/SP starter system includes an EXEC procedure that helps you load and then save this segment. CMS contains all the necessary linkage to load the CMSDOS segment when it is needed.

## *User Requirements*

In order to use discontiguous saved segments, you must:

- Allocate permanent space on a CP-owned volume to contain the saved segment.
- Assign a name to the segment and specify where it is to be stored on disk by defining an entry in the system name table (DMKSNT) with the NAMESYS macro.
- Load and save the segment. The VM/SP starter system has EXEC procedures to help you load and save the discontiguous saved segments for CMS (one EXEC procedure to load and save CMS/DOS, one for CMS/BAM, and one for CMS/VSAM and AMSERV).
- Be sure that the proper linkage for attaching and detaching discontiguous saved segments is in the operating system that needs the segment. CMS contains the linkage necessary to attach and detach the discontiguous saved segments it supports.
- Save a discontiguous saved system that is moved to a new DASD extent.

Usually, the direct access storage space is allocated and the system name table entries are created during system generation. You allocate DASD space as permanent (PERM) by executing the Format/Allocate program. This program is executed during system generation, but it is a standalone program that can be executed at any time. During system generation, you designate the CP-owned volumes by coding the SYSOWN macro of the DMKSYS file. The system name table (DMKSNT) is also created during system generation. If, at some time after system generation, you wish to change the DMKSYS or DMKSNT files, you can do a partial system generation and reassemble those files using the GENERATE EXEC procedure. GENERATE is described in the *VM/SP Installation Guide*. You can also load and save a discontiguous saved segment any time after system generation.

Notes:

1. For each shared segment specified, 64K of virtual storage is reserved. The number of pages actually saved (via the SAVESYS command) can be less than a segment. However, only one saved system name can be associated with each 64K request.

2. For each shared named system specified, page zero of the first shared segment should always be saved via a SAVESYS command.

### *Loading and Saving Discontiguous Shared Segments*

Before a discontiguous saved segment can be attached and detached by name, it must be loaded and saved. The discontiguous saved segment must be loaded at an address that is beyond the highest address of any virtual machine to which it will be attached. It is the system programmer's responsibility to make sure the name segment is loaded at an address that does not overlay the defined virtual machine or any other named segment that may be attached at the same time.

The load address for the discontiguous saved segment should be just beyond the largest virtual machine that uses it. If the load address is unnecessarily high, real storage is wasted because CP must have segment table entries for storage that is never used.

For example, assume you have five CMS virtual machines in your installation. Also assume that all five use the CMS support for DOS program development and testing which is in a 32K segment named CMSDOS. If each of your five CMS virtual machines has a machine size of 320K you should load the CMSDOS segment just beyond 320K. If you load CMSDOS at a much higher address, for example 512K, you are wasting real storage. In this case, whenever one of your CMS virtual machines attaches the CMSDOS segment, CP creates segment table entries for a 544K (512K + 32K) virtual machine. Although the virtual machine cannot refer to storage addresses beyond 320K or below 512K, CP still must have segment table entries in nonpageable real storage for those virtual addresses.

Once the named segment is loaded at the correct address, you can save it by issuing the CP SAVESYS command. To be sure that the CMS discontiguous saved segment has segment protection, set the storage key for the segment, via the CMS SETKEY command, to something other than X'F' before you save it.

The format of the CMS SETKEY command is:

| SETKEY | key systemname [startadr] |
|---|---|

*where:*

`key` is the storage protection key, specified in decimal. The valid keys are 0-15.

`systemname` is the name of the saved system or segment for which the storage protection is being assigned.

`startadr` is the starting address (in hexadecimal) at which the keys are to be assigned. The address must be within the address range defined for the saved system or discontiguous saved segments. Using the

startadr operand, you can issue the SETKEY command several times and, thus, assign different keys to various portions of the saved system or segment.

## *How the Interface Works*

The linkage to attach and detach discontiguous saved segments is supported via several CP DIAGNOSE codes.

Since the virtual machine is responsible for insuring that the discontiguous saved segment that it is attaching does not overlay other programming code, it must know how much virtual storage it has. By issuing DIAGNOSE code X'60' during its initialization process, the virtual machine can determine its virtual machine storage size.

When the virtual machine needs to attach a discontiguous saved segment, it must first ensure that the segment is available and that it does not overlay existing storage. By issuing the DIAGNOSE code X'64' with a subcode of X'0C', it can verify that a loadable copy of the discontiguous shared segment exists on a CP-owned volume. This DIAGNOSE code is called the FINDSYS function. FINDSYS returns the starting address of the segment. The virtual machine should compare the starting address of the segment to its own ending address; if the segment does not overlay existing storage, it can be loaded.

A LOADSYS function is provided by the CP DIAGNOSE code X'64' and subcodes X'00' and X'04'. The section "Diagnose Instruction in a Virtual Machine" contains a complete description of the Diagnose codes used in the discontiguous saved segment interface. If you want CMS to load the named segment in nonshared mode, you may do so by issuing the CMS command:

```
SET NONSHARE segmentname
```

before CMS attaches the named segment. If the segment is loaded in nonshared mode you can test and debug it using the CP TRACE, STORE, and ADSTOP commands and the CMS DEBUG subcommands BREAK and STORE.

When CMS loads a named segment in shared mode, it issues the CP DIAGNOSE code X'64' with subcode X'0000'. CMS also issues the same code with subcode X'0004' to load the named segment in nonshared mode.

When a discontiguous saved segment is loaded (or attached) to a virtual machine, CP expands its segment table entries for that virtual machine to reflect the highest address of the virtual machine.

When a named segment is successfully loaded, all of its storage is addressable by the virtual machine. For example, when CMS attaches a named segment, it can execute the routines contained in that segment. All of the commands that are executable for CMS are also executable for the attached named segment, with the following exceptions:

- The response for the CP QUERY VIRTUAL STORAGE command does not reflect the storage occupied by the named segment.
- If you execute a command that alters storage (such as STORE), you are given a nonshared copy of the named segment.

When the named segment is no longer needed, it can be detached. The CP DIAGNOSE code X'64' subcode X'0008', is called the PURGESYS function; it detaches named segments. When a named segment is detached, its storage is no longer addressable by the virtual machine and CP updates its segment tables. The entries for segments beyond the original virtual machine size are deleted and the associated real storage is released.

## Shared Segment Protection

Installations may optionally protect or not protect shared segments. When segments are protected, CP ensures that a virtual machine does not access a shared segment that another virtual machine has modified. When segments are not protected, CP does not provide this service.

If a virtual machine modifies an unprotected shared segment, other virtual machines sharing the segment may be affected by the modification. Therefore, before running without shared segment protection, ensure that none of the virtual machines modify shared segments.

Shared segments modified by the CP commands TRACE, ADSTOP, or STORE are handled differently by CP. In this case, CP gives exclusive use of the modified segment to the virtual machine that modified it. CP provides an unmodified copy of the segment for other virtual machines.

The VM/SP default is to protect shared segments. To turn off segment protection, use the NAMESYS macro instruction. This macro instruction can also turn on segment protection. Instructions for using the NAMESYS macro instruction are in the section "The NAMESYS Macro for Saved Systems".

When segment protection is on, CP protects segments in the following way. Before dispatching a virtual machine, CP determines if the current virtual machine altered any pages within the shared segments. If a page was altered, CP sends a message to the current virtual machine to identify the altered page, makes the altered page inaccessible, and stops the current virtual machine by placing it into console function mode. CP then dispatches another virtual machine. To resume execution on the virtual machine that CP stopped, the operator of that machine must issue the class G BEGIN command.

To make an altered page inaccessible, CP frees the storage the page occupied. Later, when a virtual machine references the page, CP brings a fresh copy of the page into storage.

Shared segment protection supports:

- The virtual machine assist feature and Extended Control-Program Support for named shared systems.
- The execution of all options of the CP STORE command in shared segments, including branch and instruction tracing.
- The execution of the CP STORE and ADSTOP commands in shared segments.
- The execution of the STORE and BREAK subcommands of the CMS DEBUG command.

CP's handling of storage keys includes the following:

- No distinction is made between shared and nonshared systems for storage key fetch instruction simulation, DISPLAY command execution, and page key handling.

- A mask in control register 6 prevents the ISK (insert storage key) and SSK (set storage key) instructions from being handled by the VMA feature. This is necessary because VMA updates the key on SSK instructions (including the SWPTABLE fields), but the new value is not detected by the hardware change bit monitoring.

CP does not permit a user of shared systems to set storage keys via the Set Storage Key (SSK) instruction. Thus, one user cannot prevent other users from accessing shared storage.

I/O activity into shared segments is monitored by channel program translators. A channel protection error occurs if a virtual machine attempts to read data into a shared segment.

The STCP command may be used to alter shared segments. When the STCP command is used to alter shared segments, the change is reflected to all users of the shared segments; the altered shared system is not assigned to the user issuing the STCP command. Whenever the STCP command is issued for a shared segment, storage is updated and the page that changed is written to the paging volume, thus reflecting the change to all users of the shared segment.

### *Virtual Machine Operation*

If you issue a STORE, ADSTOP, or TRACE command that alters a storage location within a shared segment, you receive the following message:

```
DMKVMA181E SHARED COPY SYSTEM name REPLACED WITH NON-SHARED COPY
```

Execution continues in your virtual machine; however, you are now executing your own copy of the shared system in nonshared mode. The nonshared system you are executing includes the change you just made; all other users of the shared system continue to execute in shared mode and are not affected by your change.

If you alter a shared page by any means other than the TRACE, ADSTOP, or STORE command, you receive the following message:

```
DMKVMA456W CP ENTERED; name SHARED PAGE hexloc ALTERED
```

You must enter the BEGIN command to continue execution. The altered page is returned to free storage by CP, and you may continue with an unaltered system in shared mode.

If you issue an STCP command that alters the storage of a shared segment, storage is altered and the page altered is written to the paging volume. All users, including you, remain in shared mode and the change becomes part of the shared system. If operations overlap and you issue a STCP command for a shared page that is about to be assigned to a particular user as nonshared (because he just altered it), you receive the following message:

```
DMKCDS161E SHARED PAGE hexloc ALTERED BY userid
```

You should check that you issued the STCP command correctly and then wait until the fresh copy of the saved system is loaded before reissuing the STCP command.

In attached processor systems it is invalid to issue the STCP command to a shared segment. The STORE function is not performed, and the user receives the following message:

```
DMKCDS004E INVALID HEXLOC - xxxxxx
```

## The NAMENCP Macro for 370X Control Program

The NAMENCP macro is assembled by the installation system programmer and is used to describe the location of the 370X control program. Shared segments may be specified, but they must consist of reenterable code.

When making additions, changes, or deletions to the system name table, the DMKSNT module must be reassembled. The GENERATE EXEC procedure has the facility to reassemble only the DMKSNT module. See the description of the GENERATE EXEC procedure in the *VM/SP Installation Guide.*

You must create an entry in the system name table (DMKSNT) for each unique 3704/3705 control program that you generate. If you can foresee generating several versions of the 3704/3705 control program, define extra entries in the system name table when you generate VM/SP. Use the NAMENCP macro to define 3704/3705 program entries in the system name table. See the *VM/SP Installation Guide* for information on generating the 3704/3705 control program.

### *Coding the NAMENCP Macro*

The NAMENCP macro describes the name and location of the 3704/3705 control program. See the *VM/SP Planning Guide and Reference* for the format of the NAMENCP macro.

### *Example of a NAMENCP Entry*

A DMKSNT entry to create a 3704/3705 control program could be coded as follows:

```
DMKSNTBL CSECT
EPNAME   NAMENCP CPSIZE=128K,CPNAME=EPNAME,CPTYPE=EP,          X
               SYSPGCT=10,SYSVOL=VMSRES,SYSSTRT=(010,36)       X
         END
```

In the above example, VMSRES is a count-key-data volume (IBM 3330 Disk Storage).

## Using the SAVENCP Command

Use the CMS SAVENCP command to read a 3704/3705 control program load module created by the LKED command, and to load it into virtual storage in the CMS user area. Once the load is performed, SAVENCP scans the control program image and extracts the control information required by CP. The control information is accumulated in one or more 4096-byte pages in the CMS user area. When all of the necessary control information is extracted, SAVENCP builds the Communications Controllers Parameter List (CCPARM) and issues the DIAGNOSE X'50' instruction to create the page-format copy of the control program on a CP-owned volume. The format of the SAVENCP is:

| SAVENCP | fname (ENTRY CYASTART |
|---|---|

*where:*

fname
: in this case, is the filename of the LOADLIB file where the 3704/3705 control program load resides. This name is used as the ncpname for the DIAGNOSE instruction.

ENTRY symbol
: is the external symbol of the entry point in the 3704/3705 control program load module. The standard entry for the Emulation Program is CYASTART.

The user must have a CP privilege class of A, B, or C to use the SAVENCP command. See the *VM/SP Installation Guide* for more detail of the SAVENCP command.

# The Virtual Machine Communication Facility

The Virtual Machine Communication Facility (VMCF) is part of the CP component of VM/SP. VMCF provides virtual machines with the ability to send data to and receive data from any other virtual machine.

VMCF is made up of five data transfer subfunctions, seven control subfunctions, a special external interrupt (code X'4001') to asynchronously alert virtual machines to pending messages, and an external interrupt message header to pass control information (and data, at times) to another user.

VMCF is implemented by means of subfunctions invoked using the DIAGNOSE instruction with a code of X'68' and a special 40-byte parameter list called VMCPARM. A VMCF subfunction is indicated by a particular subfunction code in the VMCPFUNC field in the parameter list.

**Note:** Before you can use any other VMCF subfunction, you must use the AUTHORIZE subfunction for communications. Before you can communicate with another user, that user must also have used the AUTHORIZE subfunction.

A special external interrupt (code X'4001') is used by module DMKVMC to notify one virtual machine of a pending transfer of data. This interrupt is also used to synchronize sending and receiving of data.

Along with this interrupt, the virtual machine receives a message header that is logged into a preassigned virtual storage area. This message header is used to define the type of request and to provide data transfer information, such as length of data. The message header is also used to notify the originator of a transaction of the success or failure of the transaction. In this case, the message header includes such information as residual counts and data transfer return codes.

Figure 6 on page 84 lists the VMCF subfunctions and gives a brief description of each. The subfunctions are described in detail in the section "Descriptions of VMCF Subfunctions".

Messages and data are directed to other virtual machines logically via the userid. Data is transferred in up to 2048-byte blocks from the sending virtual machine's storage to the receiving virtual machine's storage. The amount of data that can be moved in a single transfer is limited only by the sizes of virtual machine storage of the respective virtual machines. Use of real storage is minimal. Only one real storage page per virtual machine (a total of two pages, one for the sender and one for the receiver) need to be locked during the data transfer.

The special message facility uses VMCF to send messages from one virtual machine storage area to another virtual machine storage area. For a description of the special message facility and how it uses VMCF, see "Special Message Facility" in this section.

| Function | Code | Comments |
|---|---|---|
| AUTHORIZE | Control | Initializes VMCF for a given virtual machine. Once AUTHORIZE is executed, the virtual machine can execute other VMCF subfunctions and receive messages or requests from other users. |
| UNAUTHORIZE | Control | Terminates VMCF activity. |
| SEND | Data | Directs a message or block of data to another virtual machine. |
| SEND/RECV | Data | Directs a message or block of data to another virtual machine, and requests notification of a reply. |
| SENDX | Data | Directs data to another virtual machine on a faster but more restrictive protocol than the SEND subfunction. |
| RECEIVE | Data | Allows you to accept selective messages or data sent via a SEND or SEND/RECV subfunction. |
| CANCEL | Control | Cancels a message or data transfer directed to another user but not yet accepted by that user. |
| REPLY | Data | Allows you to direct data back to the originator of a SEND/RECV subfunction, simulating full duplex communication. |
| QUIESCE | Control | Temporarily rejects further SEND, SENDX, SEND/RECV, or IDENTIFY requests from other users. |
| RESUME | Control | Resets the status set by the QUIESCE subfunction and allows execution of subsequent requests from other users. |
| IDENTIFY | Control | Notifies another user that your virtual machine is available for VMCF communication. |
| REJECT | Control | Allows you to reject specific SEND or SEND/RECV requests pending for your virtual machine. |

**Figure 6. Virtual Machine Communication Facility (VMCF) Subfunctions**

[1]The word "Data" in this column indicates a data transfer subfunction whereas the word "Control" indicates a VMCF control subfunction.

## Using the Virtual Machine Communication Facility

The following discussion presents ideas and suggestions for using the Virtual Machine Communication Facility (VMCF).

## *VMCF Applications*

The VM/SP system with VMCF provides the user with the potential to apply new and different techniques to current applications.

### Multitasking Programming

The VMCF functions may be used to multitask virtual machines. Each virtual machine can become a subtask (parallel or otherwise) of another virtual machine. A virtual machine task can be a simple program or a large processor. The VMCF functions provide the WAIT/POST, serialization and communication facilities to control such an environment. The existing VM/SP functions provide efficient scheduling, dispatching and basic resource controls. The advantage of such an environment is that a user is less restricted by operating system (software) limitations and gains the flexibility of machine languages and hardware.

### Resource Sharing

VMCF provides a clear and concise method for sharing and serializing resources between virtual machines. The resources can range from multi-write minidisks to entire processors. The control functions for resource sharing (such as, resource management, serialization) can be contained in a virtual machine.

### Virtual Extensions to VM/SP

It is conceivable that functions could be added to VM/SP without altering the control program (CP). A special privilege class virtual machine could be used to provide additional functions to non-privilege class users using the VMCF interface. Similarly, CMS capabilities could be expanded (or at least appear to be expanded) by linking CMS with other virtual machines.

### Program Testing

The program testing capabilities offered by VMCF can range from device simulation to teleprocessing network simulation. In particular, VMCF can be used to provide external interactions from one virtual machine to another. A simulated teleprocessing network could be constructed with virtual machines. Each virtual machine would effectively become a node within the network. The network structure could range from a simple tree type structure to a complicated multi-path mesh type structure. The program logic within each node virtual machine would be the same logic as required for a real teleprocessing node. In theory, a reasonably complicated structure could be simulated without requiring the physical hardware.

The significant testing capability provided by VMCF is the ability to link the test system with test/simulation routines in another virtual machine.

### INTRA Virtual Machine Communication

Although the VMCF interface is intended for communication from one virtual machine to another it can also be used to communicate within a single virtual machine (wrap connection). The VMCF interface could conceivably be used to link one or more operating system tasks that are logically separated by the software. This would allow task to task communication rather than virtual machine to virtual machine communication.

### Virtual Multiprocessing

The VMCF interface could possibly be used to simulate a virtual multiprocessing environment.

## *Security and Data Integrity*

The VMCF interface provides the following security aids:

- The user doubleword in the external interrupt message header can be used to contain a security code to prevent unwarranted users from accessing a shared data base or other confidential information.

- The AUTHORIZE SPECIFIC option allows a user to restrict messages sent to his virtual machine. This option is useful when slave machines are to communicate only with a host machine. The slave machines can AUTHORIZE SPECIFIC with the host and prevent unwarranted users from clogging their message queues.

- The design of VMCF prevents malicious users from intercepting transactions in process for other users (for example, user D cannot execute a RECEIVE, REPLY, REJECT or CANCEL to a message sent to user B from user A).

The VMCF support module is designed such that a user is always informed of conditions that could threaten the integrity of his own data. The user is notified either with a DIAGNOSE X'68' return code or data transfer error code. There is no internal buffering of user data within the control program (CP), a message is always retained by either the SOURCE or SINK virtual machine. If a SEND type request fails, the SOURCE still has a copy of the original message. If a SINK REPLY fails, the SINK user still has a copy of the REPLY data. The Diagnose return code or data transfer error code can indicate to a user that a transaction failed. It is up to the user to preserve the associated transaction data. The following are considerations which should be noted by a VMCF user:

1. The buffer used for SOURCE data in a SEND, SENDX or SEND/RECV request should not be freed or reused until the final response external interrupt is received by the SOURCE.

2. The buffer used for SINK data in a REPLY function can be reused by the SINK after the DIAGNOSE instruction (REPLY) has successfully completed.

3. The user parameter list (VMCPARM) may be re-used upon completion of the Diagnose instruction. At that point the VMCPARM data has been copied to a VMCF control block (VMCBLOK) by the control program. A user should, however, maintain queues of VMCPARM data in order to associate an external interrupt message header (VMCMHDR) with a particular request.

4. A user should always interrogate the DIAGNOSE return code or data transfer error code for possible error conditions. It is the user's responsibility to determine the types and extent of error recovery. The DIAGNOSE return code 19 for a SOURCE SEND, SEND/RECV or SENDX request indicates that an error was associated with the SINK user and for a SINK RECEIVE or REPLY request indicates that an error was associated with the SOURCE user. The user who receives this return code does not have to invoke error recovery for himself but only be aware that the transaction did not complete successfully because of an error associated with the other user.

## *Performance Considerations*

There are several factors that can effect the performance of VMCF:

- The VMCF support module, DMKVMC, is a pageable CP module. If a user has significant paging activity, it may be advantageous to either lock the module in real storage (CP LOCK command) or alter the CP LOADLIST to make DMKVMC resident.

- It is to a user's benefit to have the user parameter list, VMCPARM, in the same 4K page as the DIAGNOSE X'68' instruction. This may eliminate a paging operation.

- User support modules using the VMCF interface should be written as reentrant modules and be contained within a CP shared segment whenever possible. This helps reduce CP paging overhead.

- For applications that involve serial message processing, the SENDX function is the most efficient. The SENDX function eliminates the need for the SINK to do a RECEIVE operation.

**Note:** Overall system VM/SP performance is not affected when VMCF is not being used by an installation.

### *General Considerations*

The SENDX function is a fast way to transfer messages or data and can be used in place of the CP MSG command where the message length exceeds the capacity of the terminal input line. Its use is somewhat restricted in that the maximum data length must be agreed upon by all VMCF users and then remains fixed unless renegotiated.

The SEND and SEND/RECV functions are better suited to transfer high volume data base type information. This type of data transfer requires the flexibility of a wide range of data lengths along with rigorous management and control techniques.

The QUIESCE function allows a virtual machine to discontinue receiving messages. The virtual machine can process those messages already stacked and then use the RESUME function to continue reception. The QUIESCE function also allows a virtual machine to process all queued messages prior to terminating VMCF operation.

The user parameter list, VMCPARM, is designed such that it can be used for any subfunction by simply varying the contents of its fields.

Users should keep copies of VMCPARMs for all requests made via the SEND, SEND/RECV, or SENDX functions. When a final response interrupt is received and the interrupt message header indicates no data transfer errors, the corresponding VMCPARM copy can be released. If a data transfer error is indicated, the copy can be used to reinitiate the transaction.

## VMCF Protocol

VMCF provides four types of protocol: SEND, SEND/RECV, SENDX, and IDENTIFY. The protocol used to communicate between two virtual machines depends on the application of VMCF and conventions established by virtual machine users authorized to use VMCF. A virtual machine must invoke the AUTHORIZE subfunction before it is allowed to use any of the other subfunctions.

The types of transactions that virtual machines can be involved in are described by a series of VMCF protocols. In these protocols the originating virtual machine is called the "source" virtual machine. The destination virtual machine is called the "sink" virtual machine.

The protocol for a transaction remains in effect for the duration of the transaction.

## *The SEND Protocol*

The SEND protocol defines a one-way transfer of data from source virtual machine storage to sink virtual machine storage. The SEND protocol uses the SEND and RECEIVE subfunctions, as described in Figure 7 . The source virtual machine first transfers data to the sink virtual machine. This is done by executing the SEND subfunction which specifies the userid of the sink virtual machine, a message ID, and the address and length of the data being sent. The sink virtual machine receives an external interrupt from CP notifying it of the data transfer request. The sink virtual machine can then respond via the RECEIVE subfunction. The RECEIVE request specifies the address and the length of the SINK buffer that is to receive the data and causes the data to be transferred from source virtual machine storage to sink virtual machine storage. When the data transfer is complete, the source virtual machine receives an external interrupt from CP, indicating that the transaction is complete and that the sink virtual machine has received the data.

All virtual machines authorized to use VMCF can send data using this protocol.

The amount of data transferred is limited only by virtual machine storage size. Data is transferred in blocks of up to 2K (when necessary) and only one real page frame is locked during the data transfer operation.

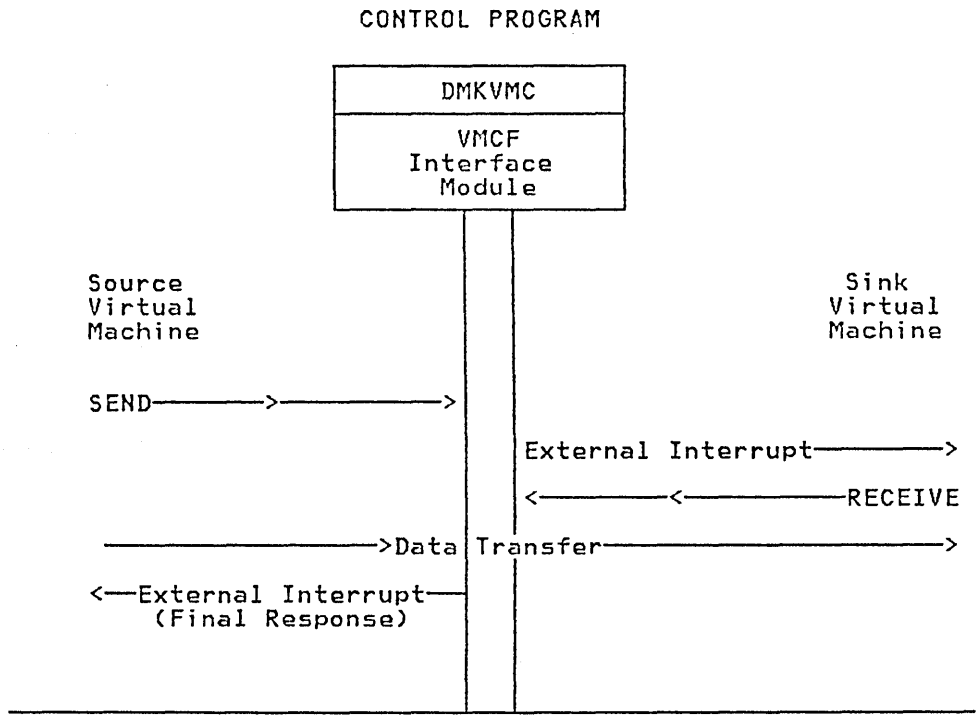



**Figure 7. The SEND Protocol**

## *The SEND/RECV Protocol*

The SEND/RECV protocol defines a transaction calling for two-way transfer of data, as described in Figure 8 The SEND/RECV protocol uses the SEND/RECV, RECEIVE, and REPLY subfunctions.

The source virtual machine initiates the transaction using the SEND/RECV subfunction. Using an external interrupt, CP notifies the sink virtual machine that there is a message waiting. The sink virtual machine uses the RECEIVE subfunction to cause the data to be transferred from the source virtual machine's storage to the sink virtual machine storage. The sink virtual machine now uses the REPLY subfunction to cause data to be transferred from its storage to the source virtual machine's storage. When the REPLY subfunction completes processing, CP causes an external interrupt in the source virtual machine, notifying it that the transaction is complete.

The SEND/RECV request requires that the source virtual machine specify the address and length of the data to be transferred and the address where data is expected from the REPLY subfunction. (Both addresses are in source virtual machine storage.) These addresses, along with the length of the data to be transferred, are specified via the VMCPARM parameter list, described below.

When RECEIVE is issued by the sink virtual machine in response to the SEND/RECV request, VMCPARM contains the address in sink virtual machine storage where data is to be received. Finally, when the REPLY request is issued, VMCPARM contains the address in the sink virtual machine storage from which data is to be transferred.

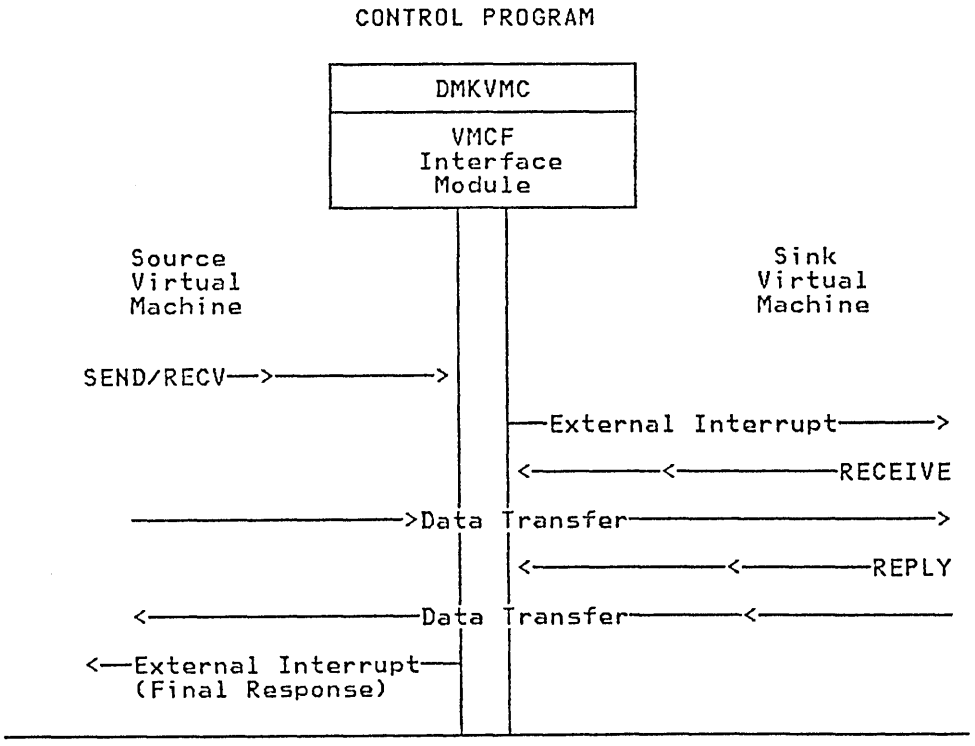



**Figure 8. The SEND/RECV Protocol**

## *The SENDX Protocol*

The SENDX protocol defines a transaction calling for an expedited one-way transfer of data. Figure 9 shows the SENDX protocol graphically. SENDX differs from the SEND protocol in that the sink virtual machine need not issue the RECEIVE subfunction; data is transferred from source virtual machine storage to sink virtual machine storage at the same time the external interrupt from CP notifies the sink virtual machine of the transaction. Data sent by the source virtual machine is placed in the external interrupt buffer of the sink virtual machine.

Virtual machines using the SENDX protocol are responsible for specifying the userid for the sink virtual machine, a message ID, the address and length of the data being sent, and the external interrupt buffer address and data length for the sink virtual machine. A virtual machine to be used as a sink virtual machine with the SENDX protocol must specify this information via VMCPARM when that virtual machine issues the AUTHORIZE subfunction. The data length specified must be at least as long as the maximum amount of data to be transferred during a transaction; it need not be limited to the usual 40-byte external interrupt buffer. Effective use of the SENDX protocol requires that VMCF users agree on a maximum size for SENDX data and then issue the AUTHORIZE subfunction with the appropriate external interrupt buffer size.

If the sink virtual machine has not provided enough SENDX buffer area in the external interrupt buffer, CP notifies the source virtual machine that the transaction was not completed.

When a SENDX data transfer is complete, CP directs a response external interrupt to the source virtual machine, notifying it that the transaction is complete.

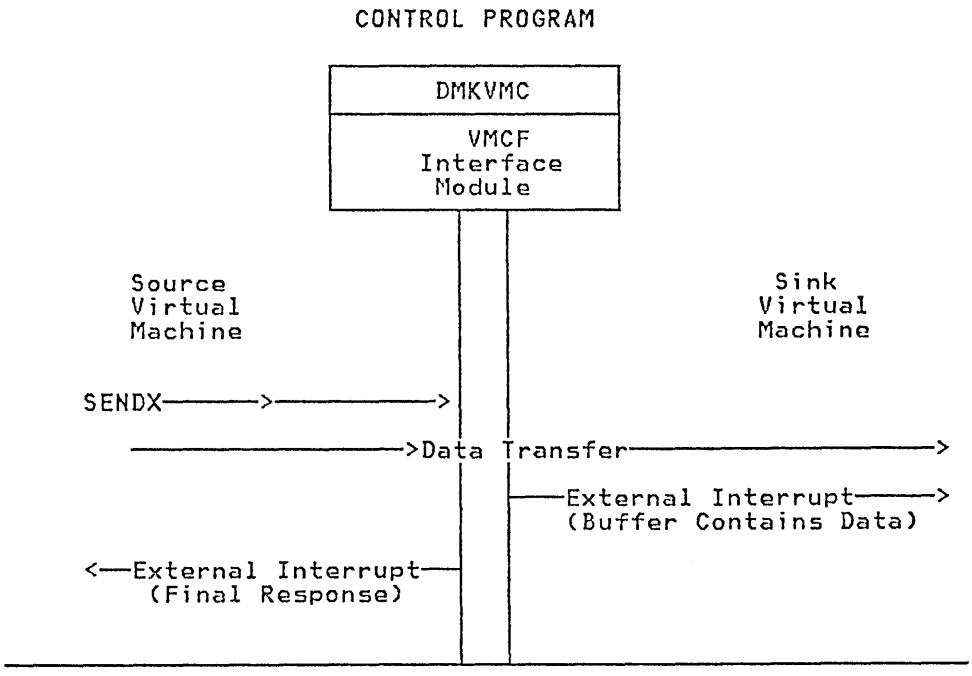



**Figure 9. The SENDX Protocol**

### *The IDENTIFY Protocol*

The IDENTIFY protocol defines a means for virtual machines to identify themselves to other virtual machines by passing user-defined control information via a standard VMCF message header. Figure &vmcf4 shows the IDENTIFY protocol graphically.

When the IDENTIFY subfunction is issued, CP directs an external interrupt to the sink virtual machine. Along with the external interrupt, the sink virtual machine receives a standard VMCF message header that contains user-defined information. The IDENTIFY protocol does not cause a response external interrupt to be directed to the source virtual machine.

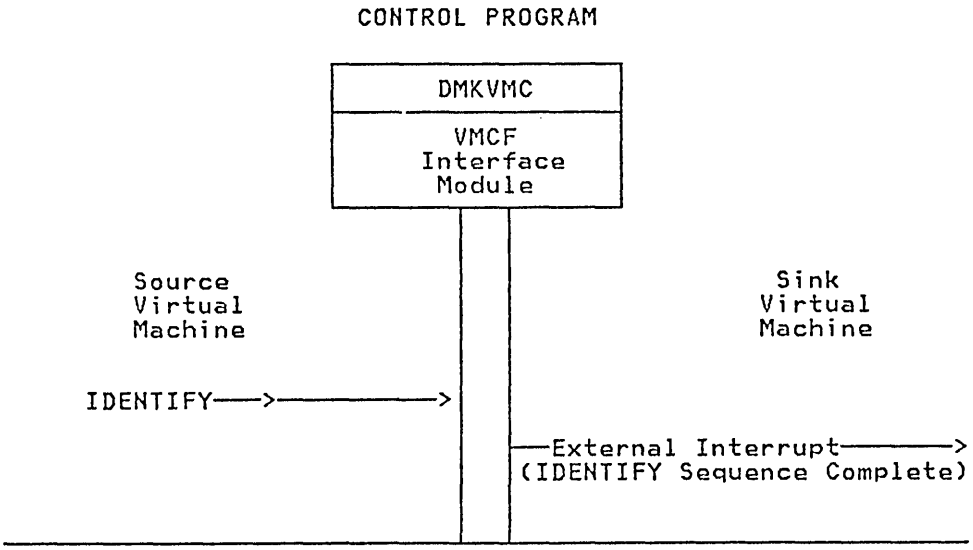



**Figure 10. The IDENTIFY Protocol**

## Descriptions of VMCF Subfunctions

There are two types of VMCF subfunctions: data transfer and control.

### *The Control Subfunctions*

The VMCF control subfunctions allow efficient management of data transfer operations from your virtual machine console. The control subfunctions are: AUTHORIZE, UNAUTHORIZE, CANCEL, QUIESCE, RESUME, IDENTIFY, and REJECT.

**AUTHORIZE: DIAGNOSE Code X'68' Subfunction Code X'0000'**

AUTHORIZE enables VMCF for a virtual machine; once AUTHORIZE has been executed, the virtual machine can execute other VMCF subfunctions and receive messages and data from other authorized VMCF virtual machines. It is possible to specify three options with the AUTHORIZE subfunction: SPECIFIC, PRIORITY, and VMCPSMSG.

The SPECIFIC option authorizes communication with a specific virtual machine. Any messages sent to the virtual machine from other than the specified virtual machine will be rejected. The SPECIFIC option can be used in an application where virtual machines desire to communicate with a master controller but not

among themselves. Under the special message facility, CP is authorized with every virtual machine that is to receive messages sent via the SMSG command. Virtual machines that are to receive messages must authorize themselves.

The PRIORITY option allows a virtual machine to authorize the receipt of priority messages. A virtual machine is allowed to send priority messages to another virtual machine only if the other virtual machine is authorized to receive priority messages. A priority message is one that is queued ahead of nonpriority messages and therefore accepted first.

When you execute the AUTHORIZE subfunction, you must specify the address and length of the external interrupt buffer for your virtual machine. The buffer must be large enough to contain a fixed message header (40 bytes). The message header identifies messages sent by other virtual machines or responses to messages you might send to your own virtual machine.

If you are going to accept SENDX-type communications, you must specify the size of the external interrupt buffer as 40 plus the maximum size of SENDX data that you plan to accept. This has the effect of authorizing SENDX protocol. That is, a virtual machine may receive data along with the external interrupt in its external interrupt buffer. When a virtual machine sends data to another virtual machine via the SENDX subfunction the data must fit in that virtual machine's external interrupt buffer or the subfunction is rejected. Messages sent via the special message facility require a buffer length of 169 bytes.

Any AUTHORIZE options in effect can be reset or changed by reexecuting the AUTHORIZE subfunction. If there are errors during execution of the AUTHORIZE subfunction, a virtual machine's authorization status is not changed.

**UNAUTHORIZE: DIAGNOSE Code X'68' Subfunction Code X'0001'**

UNAUTHORIZE terminates VMCF activity for a virtual machine. The UNAUTHORIZE subfunction causes any stacked or queued messages associated with the virtual machine to be purged. A virtual machine should execute the QUIESCE subfunction before executing UNAUTHORIZE if messages that are already queued are to be handled. When a virtual machine executing UNAUTHORIZE has pending final response external interrupts, the interrupts are purged. If a virtual machine has pending SEND external interrupts from another source virtual machine, a RESPONSE interrupt is reflected to the source indicating that the virtual machine is no longer available.

**CANCEL: DIAGNOSE Code X'68' Subfunction Code X'0006'**

CANCEL cancels a message or data transfer pending for but not accepted by another VMCF virtual machine. A virtual machine can CANCEL messages it originates with SEND, SENDX, or SEND/RECV subfunctions. A message cannot be canceled if any of the following conditions exist:

- The request was SENDX or IDENTIFY and the sink had already received the SEND external interrupt.
- The request was SEND and the sink had already executed the RECEIVE or REJECT subfunctions.
- The request was SEND/RECV and the sink had already executed the REPLY or REJECT subfunctions.

If the original request was SEND/RECV and the sink virtual machine had executed the RECEIVE subfunction but not the REPLY, the REPLY can be canceled. A virtual machine is notified of this condition with a DIAGNOSE return code. (For a description of the return codes, see Figure 11 on page 101 .)

**QUIESCE: DIAGNOSE Code X'68' Subfunction Code X'0008'**

QUIESCE temporarily rejects SEND, SENDX, SEND/RECV, or IDENTIFY requests from other virtual machines. QUIESCE allows a virtual machine to receive any stacked or queued messages but reject further SEND, SENDX, IDENTIFY, or SEND/RECV requests from other virtual machines. QUIESCE can be used to indicate to other virtual machines that the virtual machine is in QUIESCE status, authorized for communication but not able to accept messages at this time (e.g., entering slowdown, my buffers are full, try again later). The IDENTIFY subfunction could be used to inform other virtual machines that a particular user is no longer in QUIESCE status. You should execute the QUIESCE subfunction before executing the UNAUTHORIZE subfunction to avoid losing messages (see "UNAUTHORIZE: DIAGNOSE Code X'68' Subfunction Code X'0001'".) A virtual machine can reset the QUIESCE status (exit slowdown) by executing the RESUME subfunction. (See "RESUME: DIAGNOSE Code X'68' Subfunction Code X'0009'"). A virtual machine in QUIESCE status may continue to send messages to other virtual machines. QUIESCE status for a virtual machine only affects messages sent from other virtual machines.

**RESUME: DIAGNOSE Code X'68' Subfunction Code X'0009'**

RESUME cancels the QUIESCE status, allowing your virtual machine to resume reception of VMCF requests from other virtual machines. You can use the IDENTIFY subfunction to inform other virtual machines that your virtual machine is no longer in QUIESCE status. (See "IDENTIFY: DIAGNOSE Code X'68' Subfunction Code X'000A'".)

**IDENTIFY: DIAGNOSE Code X'68' Subfunction Code X'000A'**

IDENTIFY notifies another virtual machine that your virtual machine is available for VMCF communication. Use the IDENTIFY subfunction after issuing the AUTHORIZE subfunction or after your virtual machine has been in the VMCF QUIESCE state and you have issued the RESUME subfunction. IDENTIFY causes an external interrupt to be stacked for a specified virtual machine. The virtual machine executing the IDENTIFY subfunction specifies the userid of the user to receive the external interrupt. The external interrupt identifies the virtual machine executing the IDENTIFY subfunction. The IDENTIFY subfunction is provided to inform a host or controller virtual machine that a virtual machine is activated (logged on) and ready for VMCF communication. The IDENTIFY subfunction can also be used to inform other virtual machines that your virtual machine has exited QUIESCE state. There is no response external interrupt associated with the IDENTIFY subfunction.

The IDENTIFY subfunction can also be used to pass virtual machine defined control information. The fields in the VMCF parameter list (VMCPARM) not used by the IDENTIFY subfunction may be used to contain additional virtual machine data.

**REJECT: DIAGNOSE Code X'68' Subfunction Code X'000B'**

REJECT selectively rejects pending SEND or SEND/RECV requests from other VMCF virtual machines. REJECT causes a response external interrupt to be reflected to the originator of a message. The external interrupt indicates to the

originator that the message was rejected. The user doubleword within the external interrupt header may tell a user why the message was rejected. When the user of a virtual machine executes the REJECT subfunction, he specifies within the VMCF parameter list (VMCPARM) the message ID of the message to be rejected. A virtual machine cannot reject a message sent with the SENDX subfunction since the message is received at the same time the external interrupt is received. The REJECT subfunction can be executed as response to either SEND or SEND/RECV requests.

### *The Data Transfer Functions*

The data transfer operations are SEND, SEND/RECV, SENDX, RECEIVE, and REPLY. These operations involve the movement of data from one virtual machine storage to another virtual machine storage.

#### SEND: DIAGNOSE Code X'68' Subfunction Code X'0002'

SEND directs a message or block of data to another virtual machine. Specify the virtual address and length of data to be sent within the user parameter list (VMCPARM). Also, specify in the parameter list a message ID to be associated with the message and the userid of the user to receive the message (data). You can also send a doubleword of data to be transmitted within the external interrupt message header (refer to the section "VMCF User Doubleword"). If the SEND subfunction is executed with a data length of zero, only the user doubleword is transmitted to the sink virtual machine. The sink virtual machine can then respond with a RECEIVE subfunction (zero length) and pass back a doubleword of data to the source virtual machine. The external interrupt message header identifies the SEND request. When the sink virtual machine executes a RECEIVE subfunction, the message is transmitted from the source virtual machine storage to the sink virtual storage. There is no internal buffering of data within the control program (CP). All data is transferred in 2K blocks from virtual storage to virtual storage. Data is transferred in 2K blocks to test for STORE/FETCH protection violations. When the data transfer subfunction is complete, the source virtual machine receives a response external interrupt indicating that the SEND request is complete. The sink virtual machine receives a DIAGNOSE X'68' return code indicating that the RECEIVE subfunction is complete. The return code can indicate error conditions associated with the RECEIVE function or normal completion.

The sink virtual machine has the option to reject a message rather than execute the RECEIVE subfunction (See "REJECT: DIAGNOSE Code X'68' Subfunction Code X'0011'".) The source virtual machine may cancel a SEND request before the sink virtual machine has executed a RECEIVE subfunction or REJECT function (See "CANCEL: DIAGNOSE Code X'68' Subfunction Code X'0006'".)

If you are executing the SEND subfunction, you may specify the PRIORITY option. The PRIORITY option causes the external interrupt for the sink virtual machine to be queued ahead of all other nonpriority external interrupts. If there are other PRIORITY external interrupts pending for the sink virtual machine, the queuing is done in a first in first out manner. That is, PRIORITY interrupts are queued FIFO among themselves but ahead of all nonpriority interrupts.

#### SEND/RECV: DIAGNOSE Code X'68' Subfunction Code X'0003'

SEND/RECV provides the capability to both send and receive data in a single VMCF transaction. The SEND/RECV subfunction causes an external interrupt to be queued for the sink virtual machine. When the sink virtual machine receives the external interrupt, it can respond with the RECEIVE subfunction. The RECEIVE

subfunction causes data to be transferred from the source virtual storage to sink virtual storage. The sink virtual machine can then respond with a REPLY subfunction. The REPLY subfunction causes data to be transferred from specified sink virtual storage to a REPLY buffer in the source virtual storage. The source virtual machine then receives a response external interrupt indicating that the SEND/RECV request is complete.

When the source virtual machine executes the SEND/RECV function it specifies the address and length of both the SEND buffer and REPLY buffer. The address and length specifications are contained within the user parameter list (VMCPARM). The user parameter list also contains a message ID and userid of the user to receive the data (See the "VMCPARM Parameter List".)

The source virtual machine can cancel a previously executed SEND/RECV request provided the sink virtual machine has not yet executed the REPLY or REJECT subfunction. If the sink virtual machine has already executed the RECEIVE subfunction, only the REPLY can be canceled (see "CANCEL: DIAGNOSE Code X'68' Subfunction Code X'0006'").

The sink virtual machine can execute the REJECT subfunction in response to the SEND/RECV request and cause the entire operation to be terminated (See "REJECT: DIAGNOSE Code X'68' Subfunction Code X'0011'".)

The sink virtual machine can respond to a SEND/RECV request with the REPLY subfunction without executing the RECEIVE subfunction. This has the effect of informing the source virtual machine that the sink virtual machine cannot accept data but that it can send data. The source virtual machine could have executed the SEND/RECV subfunction only as a means to solicit data from the sink virtual machine. The application of this protocol is up to VMCF users. The user doubleword can be used as a means to control such an application (See "VMCF User Doubleword".)

You can execute a SEND/RECV request using the PRIORITY option. The PRIORITY option causes the sink external interrupt for the SEND/RECV request to be queued ahead of any other nonpriority external interrupts. Response external interrupts directed to the source of a PRIORITY message are also queued in priority order.

### SENDX: DIAGNOSE Code X'68' Subfunction Code X'0004'

SENDX directs data to another virtual machine via a faster but more restrictive protocol than the SEND subfunction. SENDX subfunction data reaches the sink virtual machine at the same time the SEND external interrupt reaches the sink. In order to use the SENDX subfunction, the sink virtual machine must have an external interrupt buffer large enough to contain both the standard message header and the data. The size of the external interrupt buffer is specified when you execute the AUTHORIZE subfunction. Attempts to execute SENDX are rejected when the sink virtual machine's external interrupt buffer is not large enough to contain the data. After the sink virtual machine receives the SEND external interrupt and data, a response external interrupt is directed to the source virtual machine. The SENDX subfunction eliminates the need for a sink virtual machine to execute a RECEIVE subfunction.

A SENDX request can be canceled by the source virtual machine provided the SENDX external interrupt has not yet been reflected to the sink virtual machine (See "CANCEL: DIAGNOSE Code X'68' Subfunction Code X'0006'".)

Specify the SENDX buffer address and length in the user parameter list (VMCPARM). The message ID and userid of the sink virtual machine are also specified in VMCPARM.

The SENDX subfunction can be executed with the PRIORITY option allowing the SEND external interrupt to be queued ahead of all nonpriority external interrupts for the sink virtual machine.

A SENDX request cannot be rejected by the sink virtual machine since the message is received at the same time the external interrupt is received.

You can execute the SENDX subfunction with a zero data length causing only the message header and user doubleword to be transmitted.

**RECEIVE: DIAGNOSE Code X'68' Subfunction Code X'0005'**

RECEIVE allows you to selectively accept messages or data sent via the SEND or SEND/RECV subfunctions. You must specify in the user parameter list (VMCPARM) the virtual address and length of the RECEIVE buffer. The parameter list also contains the message ID of the message to be received and userid of the virtual machine that originated the SEND or SEND/RECV request. When a virtual machine has more than one message pending, the RECEIVE function can be executed to select messages in any order by message ID.

You can execute the REJECT function in order to reject messages sent by other virtual machines. The REJECT subfunction terminates the SEND or SEND/RECV request (see "REJECT: DIAGNOSE Code X'68' Subfunction Code X'000B'".)

You can execute the RECEIVE subfunction in response to a SEND/RECV request and then execute a REJECT subfunction rather than a REPLY. The user doubleword passed back with the REJECT subfunction could indicate "RESEND", for example, if the original data was not received correctly (depending on how you want to use the protocol).

**REPLY: DIAGNOSE Code X'68' Subfunction Code X'0007'**

REPLY allows you to direct data back to the sender of a SEND/RECV subfunction. (This simulates full duplex communication.) The REPLY subfunction is used with the SEND/RECV subfunction. A user who receives a SEND/RECV external interrupt normally responds by executing the RECEIVE subfunction. The RECEIVE subfunction causes data to be transferred from the source virtual storage to the sink virtual storage. The sink virtual machine can then respond with the REPLY subfunction causing data to be transferred from specified sink virtual storage to the source virtual storage. The REPLY subfunction causes a response external interrupt to be reflected to the source virtual machine.

The user parameter list (VMCPARM) identifies the virtual buffer address and length of reply data. When the REPLY subfunction is executed, the user parameter list (VMCPARM) also contains the message ID and the userid of the virtual machine to receive the reply.

The REPLY subfunction can be executed with a zero data length indicating no response. You can transmit a reply (zero length or otherwise) using the user doubleword.

A reply can be executed in response to a SEND/RECV request without executing the RECEIVE subfunction. This indicates that you do not want to receive the message but may want to send a reply. A reply of zero length could be executed simply to terminate the SEND/RECV request. The application of the REPLY subfunction is a user decision. It must be used to terminate a SEND/RECV request, however, unless the REJECT subfunction is executed (See "REJECT: DIAGNOSE Code X'68' Subfunction Code X'0011'".) The reply is complete when the source virtual machine receives the external interrupt response.

A REPLY subfunction cannot be executed in response to a SEND request (this is a protocol violation).

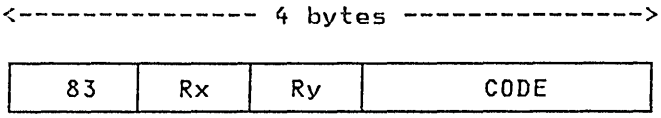
## Invoking VMCF Subfunctions

VMCF subfunctions are invoked by means of:

- DIAGNOSE code X'68' subfunction codes
- The VMCPARM parameter list
- External interrupt code X'4001'
- The external interrupt message header

### *Diagnose Code X'68'*

All VMCF subfunctions are invoked from within assembler language programs by means of DIAGNOSE code X'68':




*where:*

83 is X'83' and interpreted by the assembler as the DIAGNOSE instruction.

**Note:** There is no mnemonic for DIAGNOSE.

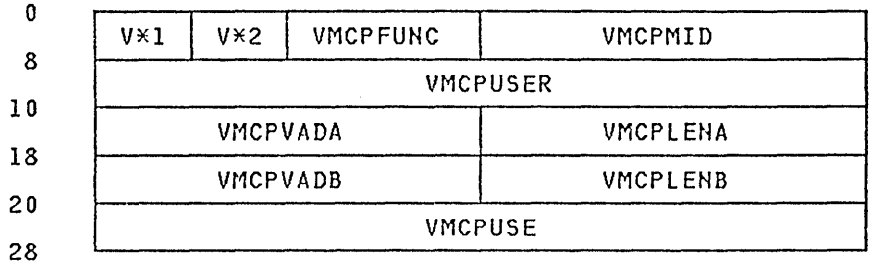
Rx specifies a register containing the address of the VMCPARM parameter list.

Ry is a register that contains a return code.

CODE is X'0068' and specifies that you are requesting execution of a VMCF.

### *The VMCPARM Parameter List*

The Rx register of DIAGNOSE X'68' contains the address of a parameter list (VMCPARM). This parameter list is used to specify the VMCF subfunction to be executed, along with other information required by VMCF to execute that function. The address of VMCPARM must be doubleword-aligned. The following is the format of the VMCPARM parameter list and a description of each of the fields in that list.




*where:*

V*1
(VMCPFLG1)
is a flag byte used to specify options associated with a particular subfunction.

This flag byte can be set to the following values:

VMCPAUTS (X'80')
Indicates, for the AUTHORIZE subfunction, an AUTHORIZE SPECIFIC request. When this bit is set, the VMCPUSER field

must contain the userid of the sink virtual machine. The status of the specified sink virtual machine is not checked by the control program (CP) at this time.

VMCPPRTY (X'40')
Indicates, for SEND, SEND/RECV, SENDX, and IDENTIFY requests, a PRIORITY message request. For an AUTHORIZE request, it indicates an AUTHORIZE PRIORITY request. You cannot send PRIORITY messages to another virtual machine unless that virtual machine has been authorized for PRIORITY messages. The SEND and RESPONSE external interrupts for a PRIORITY message are queued ahead of pending nonpriority external interrupts.

VMCPSMSG (X'20')
Indicates that the virtual machine accepts messages sent via the SMSG command.

Bits 3 through 7 are reserved for IBM use.

V*2
(VMCPFLG2)
Reserved for IBM use.

VMCPFUNC
Contains the halfword DIAGNOSE X'68' subfunction code that defines the VMCF subfunction being requested as follows:

| Command | Hexadecimal Code | Subfunction |
|---|---|---|
| VMCPAUTH | X'0000' | AUTHORIZE |
| VMCPUAUT | X'0001' | UNAUTHORIZE |
| VMCPSEND | X'0002' | SEND |
| VMCPSENR | X'0003' | SEND/RECV |
| VMCPSENX | X'0004' | SENDX |
| VMCPRECV | X'0005' | RECEIVE |
| VMCPCANC | X'0006' | CANCEL |
| VMCPREPL | X'0007' | REPLY |
| VMCPQUIE | X'0008' | QUIESCE |
| VMCPRESM | X'0009' | RESUME |
| VMCPIDEN | X'000A' | IDENTIFY |
| VMCPRJCT | X'000B' | REJECT |

VMCPMID Contains a unique message identifier associated with a transaction. The source virtual machine must originate the message ID for SEND, SEND/RECV, and SENDX requests. The message ID is used by the sink virtual machine (along with VMCPUSER) to respond to the source request with a RECEIVE, REPLY, or REJECT request. The message ID allows the sink virtual machine to selectively RECEIVE, REPLY, or REJECT messages when more than one message is enqueued. The message ID is used by both the source and sink as a unique identification for all messages. You may send messages with the same message ID to multiple users; you cannot send multiple messages with the same message ID to one user. Once a transaction is completed, however, the message ID may be reused.

VMCPUSER

Specifies the userid of the sink virtual machine for SEND, SEND/RECV, SENDX, IDENTIFY, and CANCEL requests and the userid of the source virtual machine for RECEIVE, REPLY, and REJECT requests. The sink virtual machine uses this field in combination with the message ID (VMCPMID) to respond to source requests. When the original source parameter list VMCPARM is passed to the sink as the message header VMCMHDR, the userid is changed from sink to source.

This field is also used to specify the SPECIFIC userid for an AUTHORIZE SPECIFIC request.

VMCPVADA

Contains one of four addresses, depending upon which VMCF subfunction is requested:

For SEND, SEND/RECV, and SENDX requests, VMCPVADA contains the address of the source virtual machine data. For RECEIVE requests, VMCPVADA contains the address of a sink virtual machine RECEIVE buffer. For REPLY requests, VMCPVADA contains the address in sink virtual machine storage where REPLY data is located. For an AUTHORIZE request, VMCPVADA specifies the address of the virtual machine external interrupt buffer.

The length of the associated data or buffer is specified in the VMCPLENA field.

VMCPLENA

Contains the length of the data sent by a user, the length of a RECEIVE buffer, or the length of an external interrupt buffer, whichever is specified in the field VMCPVADA. The size of the value specified in VMCPLENA is restricted only by virtual machine storage size.

The sink virtual machine can use the value in this field as the data length for RECEIVE operations.

VMCPVADB

Contains the address of a source virtual machine's REPLY buffer for a SEND/RECV request. When the sink virtual machine issues a REPLY in response to a SEND/RECV from the source virtual machine, the REPLY data is moved in this buffer. The length of the REPLY buffer is contained in the field VMCPLENB.

VMCPLENB

Specifies the length of the source virtual machine's REPLY buffer. The sink virtual machine uses this field to determine the maximum length of the REPLY. A corresponding field within the response message header contains a residual data count. The source virtual machine uses this residual count to determine the length of the sink reply. The original REPLY buffer length (less the residual count) is the length of the REPLY from the sink virtual machine.

VMCPUSE Contains the VMCF user doubleword. The user doubleword is transmitted to the sink virtual machine in the SEND message header for SEND, SEND/RECV, SENDX, and IDENTIFY requests. For

RECEIVE, REPLY, and REJECT requests, the user doubleword is transmitted to the source virtual machine within the RESPONSE message header. The sink virtual machine can transmit the user doubleword to the source virtual machine with REJECT or REPLY requests only if the original request was a SEND/RECV. The user doubleword is transmitted only with requests that result in SEND or RESPONSE external interrupts.

The following chart summarizes the VMCPARM fields required for execution of each of the VMCF subfunctions. Possible return codes associated with each subfunction are also listed. A discussion of the return codes and their meanings can be found in the section "DIAGNOSE X'68' RETURN CODES".

| VMCF Subfunction | Applicable VMCPARM Parameters | Return Codes |
|---|---|---|
| AUTHORIZE | VMCPFLG1 - SPECIFIC/PRIORITY option<br>VMCPFUNC - X'0000' - subfunction code<br>VMCPUSER - SPECIFIC userid<br>VMCPVADA - external interrupt buffer address<br>VMCPLENA - external interrupt buffer length | 0,1,2,6,15 |
| UNAUTHORIZE | VMCPFUNC - X'0001' - subfunction code | 0,2,4,15 |
| SEND | VMCPFLG1 - PRIORITY option<br>VMCPFUNC - X'0002' - subfunction code<br>VMCPMID - message identifier<br>VMCPUSER - sink userid<br>VMCPVADA - SEND data address<br>VMCPLENA - SEND data length<br>VMCPUSE - user doubleword<br><br>(See Note) | 0,1,2,4,5,8<br>9,10,15,18 |
| SEND/RECV | VMCPFLG1 - PRIORITY option<br>VMCPFUNC - X'0003' - subfunction code<br>VMCPMID - message identifier<br>VMCPUSER - sink userid<br>VMCPVADA - SEND data address<br>VMCPLENA - SEND data length<br>VMCPVADB - REPLY buffer address<br>VMCPLENB - REPLY buffer length<br>VMCPUSE - user doubleword | 0,1,2,4,5,8,9,<br>10,15,18 |
| SENDX | VMCPFLG1 - PRIORITY option<br>VMCPFUNC - X'0004' - subfunction code<br>VMCPMID - message identifier<br>VMCPUSER - sink userid<br>VMCPVADA - SEND data address<br>VMCPLENA - SEND data length<br>VMCPUSE - user doubleword<br><br>(See Note) | 0,1,2,4,5,7,8,<br>9,10,15,18 |
| RECEIVE | VMCPFUNC - X'0005' - subfunction code<br>VMCPMID - message identifier<br>VMCPUSER - source userid<br>VMCPVADA - RECEIVE buffer address<br>VMCPLENA - RECEIVE buffer length<br>VMCPUSE - user doubleword | 0,1,3,2,4,5,6,<br>12,13,15,16,17 |
| CANCEL | VMCPFUNC - X'0006' - subfunction code<br>VMCPMID - message identifier<br>VMCPUSER - sink userid | 0,2,3,4,5,11,<br>12,14,15,20 |

**Figure 11 (Part 1 of 2). VMCF Subfunctions, Parameters, and Return Codes**

| VMCF Subfunction | Applicable VMCPARM Parameters | Return Codes |
|---|---|---|
| REPLY | VMCPFUNC - X'0007' - subfunction code<br>VMCPMID - message identifier<br>VMCPUSER - source userid<br>VMCPVADA - REPLY data address<br>VMCPLENA - REPLY data length<br>VMCPUSE - user doubleword | 0,1,2,3,4,5,6,<br>12,13,15,16,17,19 |
| QUIESCE | VMCPFUNC - X'0008' - subfunction code | 0,2,4,15 |
| RESUME | VMCPFUNC - X'0009' - subfunction code | 0,2,4,15 |
| IDENTIFY | VMCPFLG1 - PRIORITY option<br>VMCPFUNC - X'000A' - subfunction code<br>VMCPUSER - sink userid<br>VMCPUSE - user doubleword<br><br>(See Note) | 0,2,4,5,9,10<br>15,18 |
| REJECT | VMCPFUNC - X'000B' - subfunction code<br>VMCPMID - message identifier<br>VMCPUSER - source userid<br>VMCPUSE - user doubleword | 0,2,3,4,12,13,15 |

**Figure 11 (Part 2 of 2). VMCF Subfunctions, Parameters, and Return Codes**

**Note:** Fields within the user parameter list that are not used by a particular subfunction may be used to contain additional user data. The data, however, can only be passed to the sink virtual machine by the source virtual machine. The REPLY buffer address and length fields (VMCPVADB+VMCPLENB) may be used to transmit additional user data for SEND and SENDX requests. All fields except VMCPFLG1, VMCPFLG2, VMCPFUNC, and VMCPUSER may be used to pass control information with an IDENTIFY request.

## *External Interrupt Code X'4001'*

External interrupt code X'4001' is a special interrupt code recognized by CP as part of a VMCF transaction. Just as virtual machines use the DIAGNOSE instruction to communicate with CP, so too CP uses this interrupt code to communicate with virtual machines. External interrupt code X'4001' and DIAGNOSE code X'68' provide the mechanism VMCF uses to synchronize message processing.

### The External Interrupt Message Header

Associated with external interrupt code X'4001' is a storage area referred to as the external interrupt message header. The external interrupt message header (VMCMHDR) contains the control information required to SEND and RECEIVE messages. The fields within the message header are, for the most part, a copy of VMCPARM parameter list fields.

Before the receiving virtual machine can receive special messages via VMCF, it must

- Enable itself to receive external interrupts
- Set bit 31 of control register 0 to a value of 1
- Authorize itself.

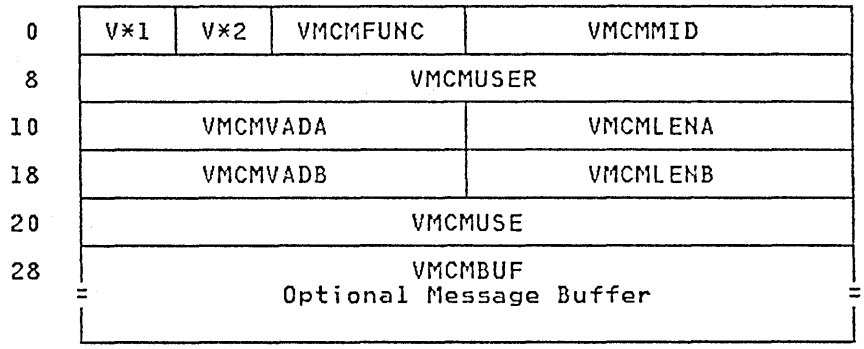
It authorizes itself by issuing DIAGNOSE Code X'68', AUTHORIZE. The parameter list, VMCPARM, specified with DIAGNOSE Code X'68' must

- Contain a pointer to an external-interrupt buffer
- Specify a buffer length of 169 bytes
- Have the special message flag (VMCPSMSG) turned on.

The receiving virtual machine may turn on this flag by setting VMCPSMSG to a value of B'1'. Optionally, the receiving virtual machine may turn on the special message flag by issuing the class G command, SET SMSG ON. For information on using DIAGNOSE Code X'68', see "Description of VMCF Subfunctions" and "Invoking VMCF Subfunctions."

CP passes the external interrupt buffer (containing the external interrupt message header) to the user's interrupt handler for processing. The user must specify the address and length of this buffer when he executes the AUTHORIZE subfunction. Then, when the user sends or receives messages, CP knows the address of the buffer and passes it to the appropriate interrupt handler routine.

Fields VMCMFUNC through VMCMUSE correspond to the fields VMCPFUNC through VMCPUSE in the VMCPARM parameter list transmitted by the source virtual machine. The format of the message header and optional SENDX data buffer is:




*where:*

V*1
(VMCMSTAT)
is a status byte associated with the message header. The bits within the status byte are defined as follows:

VMCMRESP (X'80')
Indicates final external interrupt (transaction complete). This bit is set for all RESPONSE external interrupts and the SEND external interrupt resulting from an IDENTIFY request.

VMCMRJCT (X'40')
This bit is set in a RESPONSE external interrupt to indicate that the sink virtual machine rejected the message via the REJECT subfunction.

VMCMPRTY (X'20')
This bit is set in both SEND and RESPONSE external interrupts to indicate a priority message. A virtual machine must be authorized for priority messages before it can receive them.

V*2
(VMCMEFLG)
Contains a data transfer error code indicating success or errors associated with a data transfer operation. (Refer to the section "Data Transfer Error Codes".)

VMCMFUNC
Contains the subfunction code of the original request. The sink virtual machine uses this field to determine the type of request. The possible subfunction codes are:

VMCPSEND X'0002' - SEND
VMCPSENR X'0003' - SEND/RECV
VMCPSENX X'0004' - SENDX
VMCPIDEN X'000A' - IDENTIFY

VMCMMID
Contains the message ID associated with the original source request.

VMCMUSER
Contains the userid of the source virtual machine for SEND external interrupts and the userid of the sink virtual machine for RESPONSE external interrupts. If a SMSG command was issued, "SYSTEM" appears in this field.

VMCMVADA
Contains the address of the original SEND data for SEND requests. This field would normally have no meaning to the sink virtual machine.

VMCMLENA
Indicates the length of SEND data for SEND external interrupts. It indicates a data transfer residual count for RESPONSE external interrupts.

VMCMVADB
Contains the virtual address of the REPLY buffer for SEND/RECV requests. This field has no meaning to the sink virtual machine.

VMCMLENB
Contains the length of the source virtual machine REPLY buffer for SEND/RECV external interrupts; contains the residual REPLY count for RESPONSE external interrupts. The sink virtual machine uses this field to determine the maximum length of the REPLY; the source virtual machine uses this field to determine the length of the sink virtual machine REPLY data.

VMCMUSE
Contains the user doubleword, which is transmitted to the sink virtual machine with SEND external interrupts and to the source virtual

machine with RESPONSE external interrupts. If a SMSG command was issued, this field contains the virtual machine identifier of the issuer of that command.

VMCMBUF
: This is the optional data buffer used by the SENDX subfunction. The data sent with the SENDX subfunction is moved into this buffer. The buffer size is specified when a virtual machine executes the VMCF AUTHORIZE subfunction.

## *VMCF User Doubleword*

VMCF provides a doubleword for user data that can be transmitted within the external interrupt message header. A user supplies the doubleword of data within the parameter list (VMCPARM) for certain VMCF requests (that is, SEND, SENDX, SEND/RECV, RECEIVE, REPLY, IDENTIFY, and REJECT). You can use the user doubleword in any manner you desire. The doubleword is transmitted within the external interrupt message header for both SEND and RESPONSE type external interrupts.

The user doubleword can be used for control information in a user-defined higher level protocol. That is, you could have your own message headers defined within the data transmitted from one virtual machine to another. The user doubleword could be used to control such a protocol.

The user doubleword can also be used as a security code or provide additional information for subfunctions such as IDENTIFY and REJECT. You can specify a zero data length for any VMCF transaction. The effect of this is that only the external interrupt message header with user doubleword is transmitted or received.

## *DIAGNOSE X'68' Return Codes*

The virtual machine initiating a VMCF request receives a return code in the general register specified as "Ry" in the DIAGNOSE instruction. The return code indicates successful completion of the request or error conditions associated with the request. Figure 12 is a description of all possible return codes returned to a virtual machine executing the DIAGNOSE X'68' subfunction.

| Return Code | Meaning |
|---|---|
| 0 | The normal response. Indicates successful completion of a request or successful initiation of a request. For example, for an AUTHORIZE request, 0 indicates that the AUTHORIZE function is complete; for a SEND request, 0 indicates that the SEND was successfully initiated. The SEND request, of course, would not be complete until the final RESPONSE external interrupt was received by the source virtual machine. |

**Figure 12 (Part 1 of 3). DIAGNOSE Code X'68' Return Codes**

| Return Code | Meaning |
|---|---|
| 1 | Invalid virtual buffer address or length. A virtual machine attempted to execute a VMCF subfunction but specified an invalid address or length:<br><br>• External interrupt buffer not within virtual storage.<br>• External interrupt buffer address not doubleword aligned.<br>• Message data or buffer not within virtual storage.<br>• External interrupt buffer less than the standard message header length. |
| 2 | Invalid subfunction code. A virtual machine attempted to execute a VMCF subfunction but specified an unsupported subfunction code. |
| 3 | Protocol violation. A virtual machine attempted to execute a subfunction which would violate the defined protocol:<br><br>• Cancel a message it did not originate.<br>• Reply to a message not sent via SEND/RECV.<br>• Executed more than one RECEIVE to a SEND or SEND/RECV request. |
| 4 | Source virtual machine not authorized. A virtual machine attempted to execute a subfunction (other than AUTHORIZE) but was not authorized to use VMCF (had not successfully executed the AUTHORIZE subfunction). |
| 5 | User not available. A virtual machine attempted to execute a function and specified a virtual machine currently not available for VMCF communication:<br><br>• Not logged on.<br>• Not authorized for VMCF communication.<br>• Virtual machine authorized SPECIFIC for some other virtual machine. |
| 6 | Protection violation. A virtual machine attempted to execute a VMCF function that would result in a STORE or FETCH protection violation. The virtual machine specified a data or buffer address that contained a storage key other than its current PSW key (assume key was nonzero). This return code is also set if a virtual machine attempts to receive data in a CP-owned shared segment. |
| 7 | SENDX data too large. A virtual machine attempted to execute a SENDX request but specified a SENDX data length larger than the sink virtual machine external interrupt buffer. |
| 8 | Duplicate message. A virtual machine attempted to execute a SEND-type function and specified a message ID and virtual machine userid for which there was already an active message. |
| 9 | Target virtual machine in QUIESCE status. A virtual machine attempted to execute a SEND-type function and specified a sink virtual machine userid of a virtual machine in QUIESCE status. |
| 10 | Message limit exceeded. A virtual machine attempted to execute a SEND subfunction but already had 50 messages active. The virtual machine should clear any pending RESPONSE external interrupts or CANCEL previously sent messages in order to continue processing. |

**Figure 12 (Part 2 of 3). DIAGNOSE Code X'68' Return Codes**

| Return Code | Meaning |
|---|---|
| 11 | REPLY canceled. The source virtual machine executed a CANCEL to a previous SEND/RECV request. The sink virtual machine had already RECEIVED the message but had not yet executed a REPLY. The sink virtual machine REPLY in this case is canceled. The sink virtual machine receives return code 12 (message not found) when it executes the REPLY subfunction. |
| 12 | Message not found. A virtual machine attempted to execute a subfunction and specified a message ID and virtual machine userid for a message that does not exist. The message may have existed at one time but could have been cancelled by the originator. |
| 13 | Synchronization error. The sink virtual machine attempted to respond to a message for which it had not yet received the SEND external interrupt. This condition can occur if the sink virtual machine is anticipating certain messages but does not wait for the SEND external interrupt. |
| 14 | CANCEL too late. A virtual machine attempted to CANCEL a message that had already been processed. The sink virtual machine had already responded with RECEIVE or REJECT (SEND request) or REPLY or REJECT (SEND/RECV request). This return code is also set if a virtual machine attempts to CANCEL a SENDX request for which the sink virtual machine had already received the SEND external interrupt. |
| 15 | Paging I/O error. A virtual machine attempted to execute a subfunction which resulted in an uncorrectable paging I/O error. This is a hardware failure. |
| 16 | Incorrect length. A virtual machine executed a RECEIVE or REPLY function and specified a RECEIVE buffer length less than the source virtual machine SEND data length or a REPLY data length larger than the source virtual machine REPLY buffer length. The source virtual machine receives a data transfer return code identifying the condition. |
| 17 | Destructive overlap. A virtual machine executed a RECEIVE or REPLY function and specified a RECEIVE buffer address which overlapped the source virtual machine SEND data address or a REPLY data address that overlapped the source virtual machine REPLY buffer address. This condition can occur only when a virtual machine is sending messages to itself (a "wrap connection"). |
| 18 | User not authorized for PRIORITY messages. A virtual machine attempted to send a PRIORITY message to a virtual machine that was not authorized to accept PRIORITY messages (that is, had not executed the AUTHORIZE function with the PRIORITY option). |
| 19 | Data transfer error. A virtual machine executed a request that resulted in a data transfer error condition associated with the other virtual machine. The return code is returned to the sink virtual machine to indicate that the transaction did not complete successfully. |
| 20 | CANCEL - busy. A virtual machine attempted to cancel a message being processed. If this is a SEND/RECV request and the RECEIVE subfunction is in process, repeated retries may cancel the REPLY subfunction. |

**Figure 12 (Part 3 of 3). DIAGNOSE Code X'68' Return Codes**

### *Data Transfer Error Codes*

When a virtual machine executes a SEND, SENDX, or SEND/RECV subfunction, the normal DIAGNOSE return code is zero, indicating that the request was successfully initiated. However, when the actual data transfer takes place, errors can occur. All errors occurring at data transfer time are communicated to the source virtual machine in the RESPONSE external interrupt message header, VMCMHDR. Figure &diag32. shows error codes indicating conditions that are possible after the SENDX, SEND, or SEND/RECV request is initiated. The error codes correspond to DIAGNOSE return code numbers.

| Error Code | Meaning |
|---|---|
| 0 | The normal response (no errors). |
| 1 | Invalid buffer address or length. The SEND and/or RECEIVE buffers used for a data transfer operation are not within the virtual machine's virtual storage. The beginning and ending addresses were valid when a request was initiated but all addresses are not valid. |
| 5 | User not available. The sink virtual machine executed the UNAUTHORIZE function, reexecuted the AUTHORIZE SPECIFIC subfunction, or implicitly reset his virtual machine after the source virtual machine request was initiated. |
| 6 | Protection violation. The storage key for a virtual machine's SEND or RECEIVE buffer did not match its PSW key at the time the transfer was initiated. (Assume the key was nonzero.) This error code is also set if a virtual machine attempts to RECEIVE data into a CP-owned shared segment. |
| 7 | SENDX data is too large. The sink virtual machine reexecuted AUTHORIZE and specified an external interrupt buffer size less than the buffer size at the time a SENDX subfunction was executed. The SENDX data no longer fits in the sink virtual machine buffer. |
| 15 | Paging I/O error. An uncorrectable paging I/O error occurred during the data transfer operation attempting to fetch a virtual machine SEND or RECEIVE buffer. This is a hardware failure. |
| 16 | Incorrect length. The sink virtual machine executed a RECEIVE subfunction with a data length (VMCPLENA) smaller than the original SEND data length or a REPLY subfunction with a REPLY data length larger than the source virtual machine REPLY buffer length. |
| 17 | Destructive overlap. A virtual machine was communicating with itself in a "wrap connection" and his SEND or RECEIVE buffers overlapped one another (intra-virtual machine communication). |
| 19 | Data transfer error. A data transfer error occurred which was associated with the other virtual machine. The transaction did not complete successfully. |

**Figure 13. DIAGNOSE Code X'68' Data Transfer Error Codes**

# Inter-User Communications Vehicle

The Inter-User Communications Vehicle (IUCV) is a communications facility that allows users to pass any amount of information. IUCV enables a program running in a virtual machine to communicate with other virtual machines, with a CP system service, and with itself.

An IUCV communication takes place between a source communicator and a target communicator. The communication takes place over a predefined linkage called a path. Each communicator can have multiple paths, and each communicator can receive or send multiple messages on the same path simultaneously.

IUCV provides functions, through the IUCV macro instruction, to:

- Create and dismantle paths
- Send and reply to messages
- Determine if messages are pending and describe a pending message
- Selectively receive or reject messages.

Each message is represented to CP by a control block called a MSGBLOK. This MSGBLOK is moved among different queues at different stages in a communication. Communicators can receive information about pending messages either by interrogating the queues of MSGBLOKs or by receiving an external interruption for each message.

## *IUCV Paths*

The IUCV directory control statement authorizes the establishment of a path between one virtual machine and another, or between a virtual machine and a CP system service. The number of possible paths for a communicator is limited to 65,535 (via the MAXCONN keyword of the OPTION directory statement). If a maximum number of paths is not specified in the directory, a communicator can establish a maximum of four paths. For CP system services, the maximum possible paths is 4096.

Once authorized, users establish a path when the source communicator invokes the CONNECT function and the target communicator invokes the ACCEPT function. Either communicator can terminate an established path via the SEVER function. The target communicator can also prevent the establishment of a path by invoking the SEVER function. In addition, communication over a path can be temporarily suspended when a communicator invokes the QUIESCE function; the quiesced path can be reactivated when a communicator invokes RESUME.

A single communicator can have multiple paths defined, and two virtual machines may have multiple paths between them. The communicator could be a source communicator on some of its defined paths, a target communicator on other paths, and both a source and a target communicator on still other paths. Communication over any and all paths can occur simultaneously.

Every path has two ends: the source communicator's end and the target communicator's end. Each end of a path is described by a path description. There are two path descriptions for each defined path. The source communicator has a description of the path from the source's perspective and the target communicator has a description of the same path from the target's perspective.

Each of the two path descriptions for a path has a path identification that is unique for each communicator. Path identifications are assigned by IUCV when communicators invoke the CONNECT and ACCEPT functions. When invoking IUCV functions, the source communicator identifies the path by using the source's path identification. The target communicator identifies the same path to IUCV by using the target's path identification. The only relationship that exists between a path's identifications is that the two identifiers are names for the two descriptions of the same path.

IUCV groups path descriptions for all the paths defined for a communicator into a single construct called a Communication Control Table.

## *IUCV Messages*

An IUCV communication is called a message. Communication is initiated and a message created when the source communicator invokes the SEND function. The target communicator acknowledges and accepts the message by invoking the RECEIVE function.

The target communicator can optionally request information about messages sent to it by invoking the DESCRIBE function, and can refuse a message sent to it by invoking the REJECT function. The target communicator can respond to a message via the REPLY function.

Communication is terminated and the message is destroyed when the source communicator issues the TEST COMPLETION function or handles an IUCV message complete external interrupt.

An IUCV message is represented within CP by a control block called a MSGBLOK. IUCV creates a MSGBLOK when a communication is initiated and destroys the MSGBLOK when a communication is terminated.

## *Message Queues*

During its lifetime, an IUCV message (MSGBLOK) moves among three IUCV queues. The IUCV queues are:

- Send queue - contains information about messages sent to a target communicator that the target communicator has not yet received.
- Receive queue - contains information about messages received by a target communicator that the target communicator has not yet replied to.
- Reply queue - contains information about messages replied to by a target communicator that the source communicator has not yet terminated.

IUCV moves the messages among the queues when a user issues the SEND, RECEIVE, REPLY, or TEST COMPLETION function. When a source communicator issues the SEND function, IUCV creates a message (MSGBLOK) and moves it to the target communicator's SEND queue. When the target invokes the RECEIVE function, the message is moved to the target's own RECEIVE queue. IUCV moves the message to the source communicator's REPLY queue when the target communicator invokes the REPLY function. When the source communicator issues the TEST COMPLETION function, IUCV removes the message from the REPLY queue, destroys the message, and completes the communication.

Figure 14 illustrates the movement of messages between the IUCV queues:

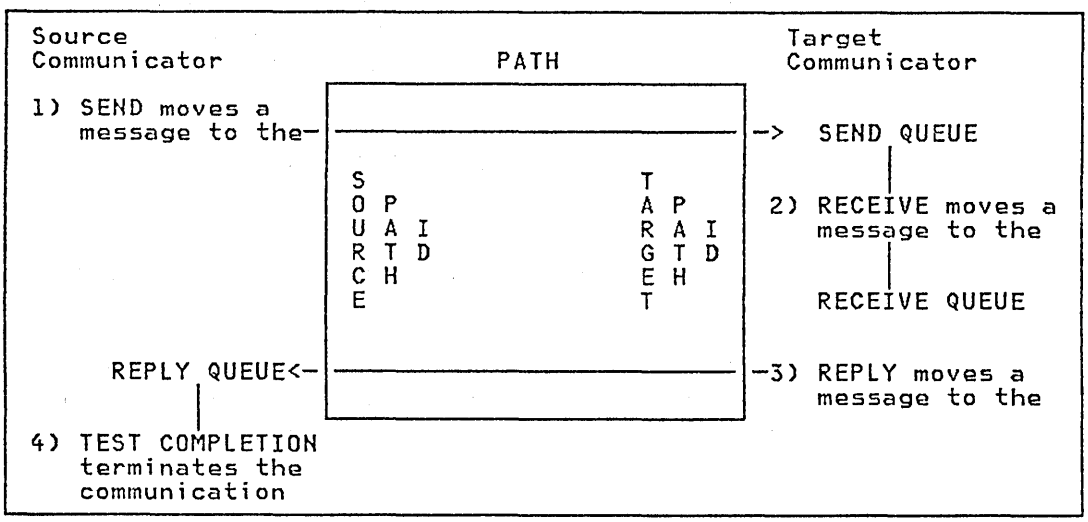



**Figure 14. IUCV Queues**

## *Message Data Transfer*

While a message (MSGBLOK) moves among the IUCV queues, IUCV moves the actual data associated with the message only twice during a complete communication. IUCV moves data when the target communicator issues the RECEIVE and REPLY functions.

IUCV moves data among four data areas during a complete communication. When the target communicator issues the RECEIVE function, IUCV moves the message data from the source communicator's SEND area to the target communicator's RECEIVE area. When the target communicator issues the REPLY function, IUCV moves data from the target communicator's REPLY area to the source communicator's ANSWER area.

Figure 15 illustrates the movement of message data during an IUCV communication.

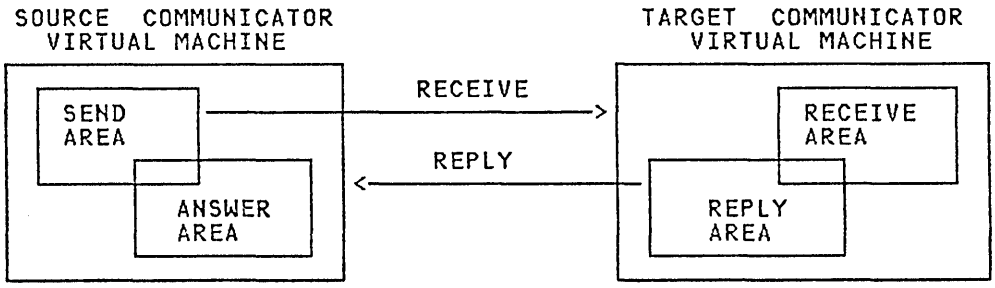



**Figure 15. IUCV Data Transfer**

The MSGBLOK representing the message contains the addresses and lengths of the source communicator's SEND and ANSWER areas. These locations may overlap.

CP performs storage protection checking for all data moved during an IUCV communication. IUCV stores the source communicator's PSW key in the MSGBLOK. When the target communicator executes a RECEIVE or REPLY function, IUCV uses the PSW key for protection checking in the source virtual machine.

IUCV uses the target communicator's PSW key at the time of the RECEIVE or REPLY to check data accesses in the target virtual machine.

### *Message Identification*

A message is fully identified to a virtual machine by values that are recorded in the MSGBLOK.

- Message identification - a single fullword value that identifies a message. IUCV assigns a message id when the source communicator invokes the SEND function. The message identification is generated by a sequential counter value and is unique for the system IPL.

- Message class - identifies the source message class and target message class. The message classes are arbitrary fullword values that the source communicator specifies when invoking the SEND function. The meaning of the message classes is agreed to in advance by the two communicators. IUCV places no restrictions on the values specified for message class. The communicators can use the message class to selectively dequeue messages.

- Path description and the target path description. IUCV assigns these path ids when a path is established via the CONNECT and ACCEPT functions.

There is no defined relationship between the values of the source and target path ids IUCV assigns, or between the message classes the source and the target communicators use. None of these values need be the same although they refer to the same message. Only the message identification has the same value for both target and source communicators.

Thus, when invoking IUCV functions, the source communicator refers to a message by a combination of its source path id, source message class, and message id. The target communicator refers to the same message by a combination of its target path id, target message class, and message id. When the target communicator issues the DESCRIBE function, IUCV provides the target's identifiers.

In addition, IUCV provides another message identifier for the source communicator. When invoking the SEND function, the source communicator may specify a message tag. IUCV does not assign a value or meaning to the tag; its use is determined solely by the source communicator. For example, the source communicator can use the message tag to tie a completed message to the original SEND request. IUCV presents the tag to the source communicator when the message completes.

Finally, a message can be identified as a priority message when the source communicator invokes the SEND function. IUCV enqueues a priority message ahead of any nonpriority messages on the target communicator's SEND queue and behind any earlier priority messages. The installation must authorize a path to handle priority messages in the IUCV directory control statement.

## *Pending IUCV Communications*

A communicator can receive notification of pending IUCV messages in two ways: by receiving external interruptions or by interrogating the SEND and REPLY queues.

### IUCV External Interrupts

To enable IUCV external interruptions, communicators must:

- Invoke the DECLARE BUFFER function to indicate to IUCV where to store data associated with an external interruption.
- Set to one Bit 7 in the virtual machine's PSW; set to one submask bit 30 of control register 0.

In addition, communicators can invoke the SET MASK function to selectively enable the virtual machine to receive external interruptions for IUCV messages, replies, and functions.

IUCV functions generate a type X'4000' external interruption. When a virtual machine in EC mode receives an IUCV external interruption, IUCV places the interruption code in locations X'86' and X'87' of the virtual machine's storage. For a virtual machine in BC mode, IUCV places the code in the external old PSW. In addition, IUCV stores an external interrupt buffer containing information about the message or IUCV function at the address specified when the communicator invoked the DECLARE BUFFER function. One field of this buffer is an external interrupt subtype that indicates why the external interrupt occurred. The possible values of this field are:

- 01 - Connection pending
- 02 - Connection complete
- 03 - Path severed
- 04 - Path quiesced
- 05 - Path resumed
- 06 - Incoming priority reply
- 07 - Incoming nonpriority reply
- 08 - Incoming priority message
- 09 - Incoming nonpriority message

See "IUCV External Interrupt Buffers" for the formats of the buffers.

A virtual machine can use the SET MASK function to selectively enable or disable external interrupts for IUCV communications. The SET MASK function has mask bits that enable or disable external interruptions for:

- Priority messages
- Nonpriority messages
- Priority replies
- Nonpriority replies
- IUCV control functions

To further divide and handle the control type interrupts, the SET CONTROL MASK function may be used on the IUCV macro. The types of control interrupts may be separately enabled and disabled. These control type interrupts are:

- Connection pending
- Connection complete
- Path severed
- Path quiesced
- Path resumed

The SET MASK function is interrogated before the SET CONTROL MASK function. If you specify that all control interrupts are disabled using the SET MASK function, then the SET CONTROL MASK settings are not interrogated. If you specify that all control interrupts are enabled using the SET MASK function, then the SET CONTROL MASK settings will be interrogated to determine how to handle the individual types of control interrupts.

After IUCV initialization and until you issue the SET MASK or SET CONTROL MASK functions, all IUCV submask bits are on, enabling all IUCV external interrupts.

### Interrogating IUCV Queues

A virtual machine can only be notified of pending CONNECT, ACCEPT, SEVER, QUIESCE, and RESUME functions by receiving an external interruption. However, a virtual machine can field incoming messages or replies either by being enabled for external interruptions, or by interrogating the SEND queue (via the DESCRIBE function) or the REPLY queue (via the TEST COMPLETION function).

IUCV also provides the TEST MESSAGE function to determine the presence of any messages on a communicator's SEND queue or REPLY queue. If no messages are present, the virtual machine goes into a wait state until a message comes in.

For example, if a source communicator sends a priority message, IUCV queues an external interruption (subtype 08) for the target communicator. If the target virtual machine is both enabled for external interruptions (bit 7 in the virtual PSW and submask bit 30 in control register zero are set to one), and enabled for priority messages (via the SET MASK function), then the target virtual machine receives an external interruption. If the target virtual machine is not enabled for external interruptions or is not enabled for priority messages, the message remains queued on the target's SEND queue. If the target virtual machine is not enabled for external interrupts or priority messages, it can issue the DESCRIBE function to obtain information about the message, and the pending external interrupt for that message is cleared. The target virtual machine can store the information and can later RECEIVE or REJECT the message.

**Note:** If a communicator is enabled for external interruptions and issues the DESCRIBE or TEST COMPLETION function, results are unpredictable. It can not be determined whether information about a particular message is received via external interruption or by the completion of DESCRIBE or TEST COMPLETION. However, IUCV supplies information about a message only once.

When a communicator has completed all communications, the virtual machine may invoke the RETRIEVE BUFFER to

- Cause IUCV to stop using the external interruption buffer created by the DECLARE BUFFER function
- Prevent further IUCV communication.

**Note:** IUCV external interruptions are not reflected to CP system code. See the section, "CP Communications" for details.

## *CP Communications*

IUCV communications with CP system services treat CP as a single virtual machine. For this reason, a distributing mechanism in IUCV (the communication processor) gathers initial information about a message and routes it to the proper module in CP for processing.

Thus, IUCV provides:

- Routing of connections from virtual machines to CP system services
- Routing of messages received via IUCV to CP system services
- Routing of REPLYs received via IUCV to the CP system service that issued the SEND
- Severing of virtual machines from system services

External interrupts are not reflected to CP system code. For communications to CP services, external interrupts are replaced with one of two possible linkages depending on whether the function was initiated outside CP or whether it was initiated from within CP. For data targeted for a CP service that was initiated in a virtual machine, there is a table of entry points which tell IUCV where to pass control. For replies targeted for a CP system service, virtual machines use a control block called an IXBLOK. The structure and use of an IXBLOK is similar to a CPEXBLOK.

Each CP system service that interfaces with virtual machines is uniquely defined to the IUCV communication processor. For each CP service defined to use IUCV communications, there are five entry points that can gain control from IUCV:

- One to get control for incoming connections
- One to get control for incoming messages
- One to get control when a connection to the particular service is severed
- One to get control when a QUIESCE is issued for a path
- One to get control when a RESUME is issued for a path

When any one of these entry points is given control, Register 1 points to a buffer. This buffer contains the same information in the same format as an IUCV external interrupt buffer used in virtual machine to virtual machine communications.

The CP system services that IUCV supports at this time are Console Communications Services, the Message System Service, and the DASD Block I/O System Service. To establish communications with the Console Communications Services facility, specify *CCS when invoking the CONNECT function of IUCV. To establish communications with the Message System Service, specify *MSG when invoking the CONNECT function. To establish communications with the DASD Block I/O System Service, specify *BLOCKIO when invoking the CONNECT function of IUCV. The name of a CP system service id must begin with an asterisk (*).

### *Second Level Support*

An SCP that supports IUCV communications functions correctly in a virtual machine generated by a CP system that supports IUCV.

The IUCV macro instruction generates an operation exception in the real hardware.

When a virtual machine invokes an IUCV function, it must be in a virtual supervisor state.

A virtual machine must invoke the DECLARE BUFFER function before other IUCV functions except the QUERY function. Failure to do so causes an operation exception to be reflected to the virtual machine.

Thus, an SCP can support IUCV in a virtual machine exactly as it does on real hardware.

CP system code invokes IUCV functions through a CALL linkage rather than the IUCV macro instruction.

### *Trace Table Entries*

IUCV support generates a trace table entry for *each* IUCV function. There is one trace table entry type for IUCV entries (X'15'). Each entry contains a subtype field to indicate the exact IUCV function a communicator invoked.

Whether invoked from a virtual machine or from CP system code, *all* uses of IUCV are recorded in the CP trace table. The address portion of the old PSW is recorded as part of the entry. A bit in the flags byte indicates whether this address is a real address (when invoked from CP) or a virtual address (when invoked from a virtual machine). For virtual machine addresses, the address of the associated VMBLOK can be obtained from preceding trace table entries.

The IUCV trace facilities can be suppressed at assembly time by setting &TRACE(9) to 0 or at execution time by setting the X'80' bit to 0 in TRACFLG3 in PSA.

IUCV functions invoked by other functions are also recorded as if they had been invoked from CP. These secondary functions include:

- The RETRIEVE BUFFER function generates a SEVER for all established paths.
- The SEVER function generates a REJECT for each incoming outstanding message and a PURGE for each outgoing outstanding message.
- A CONNECT issued to a CP system service passes control to that service. The selected CP system service usually invokes the ACCEPT function.
- The CP dispatcher invokes the DESCRIBE and TEST COMPLETION functions to dequeue messages intended for the CP system.

### *Audit Trail*

IUCV maintains an audit trail for each message. The audit trail is a bit significant value that records the status of the message. The value is maintained in the MSGBLOK that represents the message. The audit trail is presented to the source communicator during execution of the PURGE and TEST COMPLETION functions and when the source receives a message-complete IUCV external interrupt.

The audit trail for a message indicates:

- If the message caused a protection or addressing exception on the source communicator's send or answer buffer
- If the message caused a protection or addressing exception on the target's receive or reply buffer
- If a reply was too long for the source's reply buffer
- If a message was rejected by the target
- If a path was severed

### *Restrictions*

The following areas of IUCV are limited:

- The use of IUCV is supported for a second level CP system. The IUCV functions are not simulated, but are reflected to the second level system.
- Each virtual machine is limited to less than 65,536 outstanding connections at one time.
- IUCV does not recognize anything smaller than a virtual machine. If two communicators choose to establish multiple communication paths, it is the responsibility of these communicators to manage these paths.
- A CP system service cannot establish communication with itself.
- CP system services are limited to a total of 4,096 outstanding connections.

### *Security Considerations*

Installations control the use of IUCV through the virtual machine directory entries. If the installation has not authorized a user for IUCV communications in the directory, all requests for IUCV communications to virtual machines other than his own are denied. The installation must specifically authorize each virtual machine which is to communicate with a CP system service.

IUCV moves data from one virtual machine address space to another. At no time does a virtual machine have access to the storage or registers of CP or another virtual machine. When the user invokes the RECEIVE or REPLY functions, the data to be moved is described by a starting address and a length. The exact length specified is the maximum amount of data moved. There are no requirements placed on a virtual machine as to the location of these buffers.

IUCV assigns path ids and records the path id in each communicator's communication control table (CCT). IUCV sets up one CCT for each virtual machine and one for the CP system. A given communicator can reference only the paths recorded in its own CCT. Other references are not possible.

IUCV assigns the message id for each message. Although this message identifier may be reused, at any given time, it identifies only one message. IUCV does not use this identifier as a direct reference, but only as an operand in a comparison. It is conceivable that a virtual machine could generate a valid message identifier and use this to request a message. However, when a message id is used to request a message, a user must also specify a message class and a path id. If the specified message is not associated with the specified path id, and message class, the user cannot access the messages. If the message id, path id, and message class do match, the user could legitimately access it by specifying simply path id and/or message class without the generated message id.

The installation can limit the number of connections for a particular virtual machine in the virtual machine directory.

### *Performance Considerations*

The overhead involved in reflecting IUCV external interrupts to the virtual machine can be reduced if the buffer declared on the DECLARE BUFFER function is entirely within one page. Overhead can be reduced further if the buffer is entirely within page 0 of the virtual machine.

Modules DMKIUA and DMKIUE can be made resident to improve the performance of IUCV.

## Using IUCV Functions

Communicators invoke all IUCV functions through the IUCV macro instruction. When using the IUCV macro instruction, communicators specify which function they wish to perform. Most functions also require the address of a parameter list to contain inputs to and outputs from the requested function. Communicators can store inputs directly in the parameter list or they can specify inputs with keyword parameters. IUCV moves the values specified on the keyword parameters into the specified parameter list. For details on how to use the IUCV macro, see the section "Invoking IUCV Functions".

The following list describes the IUCV functions in the order that they might be used in a typical communication.

- QUERY - Use the QUERY function to determine how large a buffer IUCV requires to store external interrupt information. IUCV returns the number of bytes required in general register zero. In addition, use the QUERY function to determine the maximum number of communication paths that can be established for your virtual machine. IUCV returns the maximum number of paths in general register one. The QUERY function does not use a parameter list. CP system code cannot use the QUERY function.

- DECLARE BUFFER - Use the DECLARE BUFFER function to specify the address of a buffer into which IUCV can store external interrupt information. If a virtual machine receives an IUCV external interruption, IUCV stores in

this buffer information about the message, reply, or control function that caused the the interruption. Each virtual machine must declare a buffer prior to establishing any connections.

**Note:** When a communicator invokes the DECLARE BUFFER function, IUCV automatically enables the virtual machine for all five types of IUCV external interrupts. Use the SET MASK function to change these initial settings. CP system code does not declare a buffer.

- CONNECT - Use the CONNECT function to request the establishment of a communications path with another communicator. When a source communicator invokes the CONNECT function, IUCV establishes a pending connection. The path is not complete until the target communicator invokes the ACCEPT function.

- ACCEPT - Use the ACCEPT function to respond to a pending connection. When a target communicator invokes the CONNECT function, IUCV completes the connection and enables the path for use. A target communicator can refuse a pending connection by invoking the SEVER function.

- SEND - Use the SEND function to initiate a communication with another virtual machine or CP system service. When a source communicator invokes the SEND function, IUCV creates a MSGBLOK for the message and enqueues it on the target communicator's SEND queue. The message text is not transmitted to the target virtual machine until the target communicator invokes the RECEIVE function. If the installation has authorized the path for priority messages, you may indicate that the message is a priority message. IUCV queues priority messages ahead of nonpriority messages on the target communicator's SEND queue (and after any priority messages that have not yet been received). In addition, you may specify that a message is a one-way communication. When the target communicator receives a one-way communication, he cannot send a reply.

- DESCRIBE - Use the DESCRIBE function to determine the presence of any messages on the SEND queue that have not been previously described or reflected in a message-pending IUCV external interruption. If a previously undescribed and unreflected MSGBLOK is on the SEND queue, IUCV returns pertinent information about the MSGBLOK in the parameter list. The MSGBLOK description stored by IUCV consists of the path id, the target message class, the message id, the message flags, the length of the message, and the length of the source's answer area. This information allows the target communicator to receive the message using the RECEIVE function. IUCV describes a particular message once. It is the responsibility of the target communicator to remove described messages from the SEND queue. Messages can be removed by invoking the RECEIVE or REJECT function. The DESCRIBE function clears the pending-message external interruption for the described message. CP system code (outside of IUCV support) cannot use the DESCRIBE function.

- RECEIVE - Use the RECEIVE function to accept messages sent via the SEND function. When a target virtual machine issues the RECEIVE function, IUCV moves the actual message data from the source virtual machine's send area to the target virtual machine's receive area. If the complete message has been moved from the send area to the specified receive area, IUCV moves the MSGBLOK for the specified message from the SEND queue to the RECEIVE queue. If the receive area cannot completely contain the message, the

MSGBLOK remains on the SEND queue and the length of the remaining data is stored in the parameter list. The target virtual machine can obtain the remainder of the message with a subsequent RECEIVE. The RECEIVE function completes a one-way communication. When invoking the RECEIVE function, you can identify the message you wish to receive. Identify the message completely by specifying the message id, path id, and target message class. If you do not specify the message id, you can identify the message by path id, target message class, or both. If you do not specify any identifiers when invoking the RECEIVE function, you receive the first message that has not been partially received. Note that if a message has been partially received, you must identify the message completely to receive the remainder.

- REPLY - Use the REPLY function to respond to a message sent by a source communicator. When a target virtual machine invokes the REPLY function, IUCV moves the MSGBLOK for the specified message from the target communicator's RECEIVE queue to the source communicator's REPLY queue. Data in the target's reply area is moved to the source communicator's answer area. The target communicator can specify that a reply is a priority reply. IUCV queues a priority reply ahead of any nonpriority replies and after any earlier priority replies. When invoking the REPLY function, you must identify completely the message to which you wish to reply. Identify the message completely by specifying the message id, path id, and target message class.

- TEST COMPLETION - Use the TEST COMPLETION function to determine if any messages have been completed. When a source virtual machine invokes the TEST COMPLETION function, IUCV removes the MSGBLOK representing the specified message from the REPLY queue and destroys that MSGBLOK. When invoking the TEST COMPLETION function, you may identify which message you wish to complete. You can identify the message completely by message id, path id, and source message class. If you do not specify the message id, you can identify the message by path id, source message class, or both. If you do not specify any identifiers when invoking the TEST COMPLETION function, IUCV completes the first message on the REPLY queue. CP system code (outside of IUCV support) cannot use the TEST COMPLETION function.

- TEST MESSAGE - Use the TEST MESSAGE function to determine whether any messages or replies are pending on a communicator's SEND queue or REPLY queue. When a virtual machine invokes the TEST MESSAGE function, the virtual machine enters a wait state if neither messages nor replies are pending. If an IUCV message or reply becomes pending while the virtual machine is in the wait state, the virtual machine begins execution by re-executing the TEST MESSAGE function (which returns a condition code). By using the TEST MESSAGE function, a virtual machine avoids the necessity of external interrupt handling.

- REJECT - Use the REJECT function to refuse a specified message sent by a source communicator. After invoking the DESCRIBE or RECEIVE function, a target communicator can choose not to process a message. The REJECT function moves the MSGBLOK representing the specified message from the target's SEND queue or RECEIVE queue to the source communicator's REPLY queue. IUCV updates the message's audit trail to indicate that the message has been rejected. No message data is moved when the REJECT function is invoked. When invoking the REJECT function, you must identify which message you wish to reject. You can identify the message completely by

specifying the message id, path id, and target message class. If you do not specify the message id, you must identify the message by path id, target message class, or both.

- PURGE - Use the PURGE function to terminate a specified message sent to a target virtual machine. If the source virtual machine purges a message before the target has described or received it, the target is never aware that the message was sent. If the message is already on the source's REPLY queue, IUCV terminates the message immediately. If the message has been described to the target, IUCV notifies the target that the message has been purged. IUCV indicates that the message has been purged when the target issues the RECEIVE or REPLY function for the message. IUCV then destroys the message. When invoking the PURGE function, you must identify which message you wish to purge. You can specify only a path identifier, or a path id, message identifier, and message class. If you do not specify a message identifier, the message class is optional.

- SET CONTROL MASK - Use the SET CONTROL MASK function to enable or disable external interrupts for the IUCV control functions: connection pending, connection complete, path severed, path quiesced, and path resumed. A virtual machine must first be enabled for external interruptions by setting both bit 7 in the virtual PSW and submask bit 30 in control register zero to one. The SET MASK IUCV control bit must also be set on or the SET CONTROL MASK settings are ignored. The SET CONTROL MASK function cannot be used from CP system code.

- SET MASK - Use the SET MASK function to enable or disable IUCV external interruptions for priority messages, nonpriority messages, priority replies, nonpriority replies, and IUCV control functions. A virtual machine must also be enabled for external interruptions by setting both bit 7 in the virtual PSW and submask bit 30 in control register zero to one. The SET MASK function cannot be used from CP system code.

- QUIESCE - Use the QUIESCE function to temporarily suspend incoming messages on an IUCV path. A communicator may reactivate a path by invoking the RESUME function or may leave the path quiesced, making it a one-way path.

- RESUME - Use the RESUME function to restore communications over a previously quiesced path.

- SEVER - Use the SEVER function to reject a pending connection or to terminate a completed IUCV path. If the path is complete, both communicators must issue the SEVER function for the path to be terminated. After one communicator invokes the SEVER function, all messages outstanding on the path are terminated and IUCV notifies the communicating partner (via a SEVER external interruption). The communicating partner then can dequeue and process the terminated messages if it chooses. The communicating partner invokes the SEVER function when it finishes processing messages on the path. If the path is a pending connection, either communicator may invoke the SEVER function. If the originator of the connection invokes SEVER and the target has received the pending-connection external interruption, the target must also invoke the SEVER function. If the target invokes SEVER first, the originator must do so as well.

- RETRIEVE BUFFER - Use the RETRIEVE BUFFER function to terminate all outstanding messages and communications paths, and to end IUCV communications. CP system code (outside IUCV support) cannot use the RETRIEVE BUFFER function.

### Virtual Machine to Virtual Machine Communication

Figure 16 illustrates the sequence of functions invoked when a virtual machine communicates with another virtual machine. The functions include initialization, connection to another virtual machine, sending and receiving messages, replying to and waiting for messages, severing communications with the other virtual machine, and termination.

**Virtual Machine X Communicating to Virtual Machine Y**

| *(VIRTUAL MACHINE X)* | *(VIRTUAL MACHINE Y)* |
|---|---|
| 1 DECLARE BUFFER | 1 DECLARE BUFFER |
| 2 CONNECT to Y | |
| | 3 Get External Interrupt |
| | 4 ACCEPT |
| 5 Get External Interrupt | |
| 6 SEND to Y | |
| | 7 Get External Interrupt |
| | /or/ |
| | DESCRIBE |
| 8 TEST COMPLETION | |
| | 9 RECEIVE |
| | 10 REPLY |
| 11 Get External Interrupt | |
| /or/ | |
| TEST COMPLETION | |
| 12 SEVER | |
| | 13 Get External Interrupt |
| | 14 SEVER |
| 15 RETRIEVE BUFFER | 15 RETRIEVE BUFFER |

**Figure 16. Sequence of Functions**

1. Virtual machine X wishes to communicate with virtual machine Y. Both virtual machines must independently invoke the DECLARE BUFFER function. The buffer is used to provide the virtual machine with information about incoming external interrupts concerning IUCV functions.

2. Virtual machine X invokes the CONNECT function, indicating Y as the target. IUCV checks the directory to determine if this connection is authorized. If it is, IUCV queues an external interrupt for Y indicating that there is a pending connection for it. IUCV returns control to X at the next instruction after the CONNECT; a return code indicates that a partial connection has been established.

3. The external interrupt queued by step 2 is reflected to Y indicating a pending connection. IUCV places the external interrupt information in the buffer that Y provided in step 1. IUCV passes control to the external interrupt handler of Y.

4. Virtual machine Y interprets the external interrupt and responds with an ACCEPT to complete the connection. IUCV then completes the connection and queues a connection-complete external interrupt for X. IUCV returns control to Y at the next instruction after the ACCEPT; a return code indicates that the connection is complete.

5. The external interrupt queued by step 4 is reflected to X, indicating that the connection is complete and the communication path is available for use. IUCV places the external interrupt information in the buffer that X provided in step 1.

6. Virtual machine X issues a SEND. The SEND function queues an external interrupt for Y indicating that there is a message pending. Control returns in X at the next instruction after the SEND; a return code indicates that the message has been sent.

7. If virtual machine Y is enabled for external interrupts and for IUCV messages (via SET MASK), the external interrupt queued by step 6 is reflected to Y, indicating that a message is pending. IUCV places external interrupt information in the buffer specified in step 1. IUCV passes control to the external interrupt handler of Y. If virtual machine Y is disabled for external interrupts or IUCV messages and invokes the DESCRIBE function, IUCV places the information identifying the message in the DESCRIBE parameter list and the pending-message external interrupt for this message is cleared. IUCV passes control to the next instruction after the DESCRIBE.

8. While virtual machine Y is processing the message, virtual machine X can decide to check if the communication has been completed by issuing the TEST COMPLETION function. The condition code indicates that (in this example) the communication is not complete.

9. With the message description from step 7, virtual machine Y starts to process the message and issues a RECEIVE. The parameter list associated with RECEIVE specifies where the message data is stored in virtual machine Y.

10. When processing the message is complete, virtual machine Y responds to X by invoking the REPLY function. The REPLY function queues an external interrupt for X indicating that there is a reply pending. Control returns to Y at the next instruction after the REPLY; a return code indicates that the reply has been transferred.

11. If virtual machine X is both enabled for external interrupts and enabled for IUCV replies, the external interrupt queued by step 10 is reflected to X, indicating a reply pending. To identify the reply, the external interrupt information is placed in the buffer specified in step 1. IUCV passes control to the external interrupt handler of X. If virtual machine X is disabled for external interrupts and issues a TEST COMPLETION, IUCV places the information identifying the reply in the TEST COMPLETION parameter list and clears the queued external interrupt concerning this reply. IUCV passes control to the next instruction after the TEST COMPLETION.

12. Virtual machine X has now completed its communications with virtual machine Y and issues a SEVER to break the communications path. The SEVER function queues an external interrupt for Y indicating that the communication link has been broken. Control returns in X at the next instruction after the SEVER; a return code indicates the path has been broken.

13. The external interrupt queued by step 12 is reflected to Y indicating that the path has been broken by virtual machine X. Virtual machine Y can now do any clean up needed in its storage.

14. After virtual machine Y has completed processing, the virtual machine issues a SEVER to notify IUCV that is also is finished with the communication link. IUCV can then clean up the control blocks.

15. When all communications are complete and all communication paths have been severed, both virtual machines independently invoke the RETRIEVE BUFFER function.

### *IUCV Communications Using Parameter List Data*

To better understand how data specified in the parameter list is handled, the IUCV functions are covered in a typical user scenario:

1. The IUCV DECLARE BUFFER, CONNECT, and ACCEPT sequence must be invoked to establish the user's external interrupt buffer and a path to the target virtual machine (or CP). If you expect to receive data in the parameter list, you must authorize such communication on the CONNECT or ACCEPT by specifying PRMDATA=YES. The external interrupt information to the target communicator includes a bit indicating if PRMDATA=YES was chosen.

2. Issue an IUCV SEND request. When the data is to be passed in the parameter list, the DATA=PRMMSG option is used on the IUCV macro, and the PRMMSG= option is used to move the data into the parameter list. Or you can avoid using the macro options by initializing the parameter list yourself. The sender of the message should be prepared to handle a return code indicating that DATA=PRMMSG is not allowed if the target communicator has not specified PRMDATA=YES at connection time. A message block (MSGBLOK) is created to represent the message within CP and contains the message data until presented to the target. The message is queued on the target send queue.

3. If the target is enabled for IUCV pending-message external interrupts, the target virtual machine receives an IUCV pending-message external interrupt as a result of the SEND request in the previous step. The message data is stored in the external interrupt buffer. A flag is set in the IPFLAGS1 field of the buffer to indicate that the data is in the parameter list. Since the message data has been presented to the target, the target does not have to issue an IUCV RECEIVE for this message. If the message was a one-way message, the MSGBLOK is destroyed and the communication is complete. There is no asynchronous return of message completion given to the source (sending) virtual machine on a one-way message.

4. If the target is disabled for IUCV pending-message external interrupts and issues the IUCV DESCRIBE or RECEIVE functions, the message data is stored in the parameter list. A flag is set in the IPFLAGS1 field of the parame-

ter list to indicate that the data is in the parameter list. Since the message data is presented to the target on a DESCRIBE, the target does not have to issue an IUCV RECEIVE for this message. If the message was a one-way message, the MSGBLOK is destroyed, and the communication is complete. There is no asynchronous return of message completion given to the source (sending) virtual machine on a one-way message.

5. If the communication in the previous steps was a two-way message, a REPLY is issued by the target virtual machine. When the REPLY data is to be passed in the parameter list, the DATA=PRMMSG option is used on the IUCV macro, and the PRMMSG= option is used to move the data into the parameter list. Or, you can avoid using the macro options by initializing the parameter list yourself. The REPLYer of the message should be prepared to handle a return code indicating that DATA=PRMMSG is not allowed if the source communicator has not specified PRMDATA=YES at connection time. The message block (MSGBLOK) contains the message data until presented to the source communicator. The message block is queued on the sender's reply queue.

6. If the source communicator is enabled for IUCV message complete external interrupts, the source virtual machine receives an IUCV message-complete external interrupt as a result of the REPLY in the previous step. The message data is stored in the external interrupt buffer. A flag is set in the IPFLAGS1 field of the buffer to indicate that the data is in the parameter list. The MSGBLOK is destroyed and the communication is complete.

7. If the target is disabled for IUCV message-complete external interrupts, and issues the IUCV TEST COMPLETE function, the message data is stored in the parameter list. A flag is set in the IPFLAGS1 field of the parameter list to indicate that the data is in the parameter list. The MSGBLOK is destroyed, and the communication is complete.

8. SEVER and RETRIEVE BUFFER cause any pending messages (MSGBLOK) to be destroyed for that virtual machine. Since no asynchronous message-complete interrupt is returned to the source communicator, for one-way messages using the DATA=PRMMSG option, the source communicator must realize upon receiving an IUCV SEVER external interrupt from the target communicator, that messages may not have been received by the target.

### *Invoking IUCV Functions*

Invoke all IUCV functions through the IUCV macro instruction. In general, specify the name of the IUCV function you wish to perform, the address of a parameter list to contain input to the function, and keyword parameters. IUCV moves the values specified on the keyword parameters into the specified parameter list. Most functions require a parameter list as input to the IUCV macro instruction. Use the PRMLIST= parameter to specify the address of the parameter list. The parameter list must begin on a doubleword boundary or a specification exception results. When invoked from a virtual machine, specify the address of the parameter list as a guest real address (that is, it must be an address that is real to the virtual machine). When invoked from CP system code, the address of the parameter list must be a real address.

Supply input to IUCV functions in two ways:

- By coding keyword parameters on the IUCV macro instruction. IUCV stores values in the function parameter list based on values you specify on the macro.

- By storing required input to the function in the function parameter list before invoking the IUCV macro instruction. To store input in an IUCV parameter list, use labels generated by the IPARML mapping macro.

You may use a combination of these methods to supply input to a single IUCV function. If you specify any optional parameters on the IUCV macro, you are responsible for providing the USING for the IPARML DSECT when the macro is invoked. If you do not specify an optional parameter to initialize the parameter list, the macro assumes that you have stored a value in the parameter list prior to invoking the IUCV macro.

One advantage of using the IUCV macro instruction is that IUCV provides extensive error checking of parameter combinations when input is supplied on the macro. Many invalid parameter combinations can be detected by IUCV when you assemble the program.

You can specify several parameters either as relocatable labels or a register specification. Specify these parameters in one of the following ways:

- An addressable label in a program
- A label in the IPARML DSECT
- A register number in parentheses - (register)
- An explicit base-displacement notation -- displacement (register).

Figure 17 shows the format of the IUCV macro:

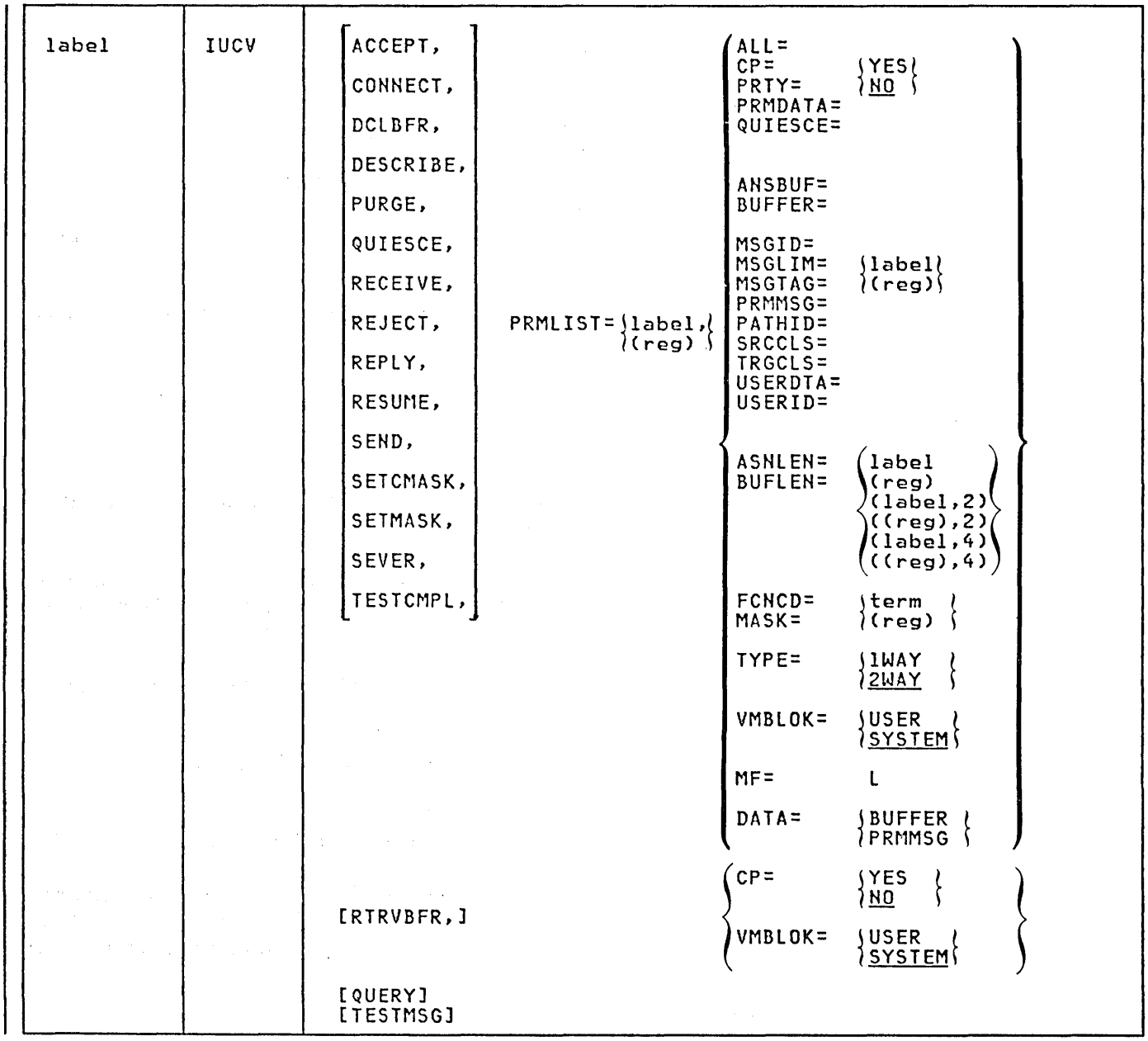



**Figure 17. IUCV Macro Instruction Format**

**where:**

| | |
|---|---|
| ACCEPT | is the ACCEPT function |
| CONNECT | is the CONNECT function |
| DCLBFR | is the DECLARE BUFFER function |
| DESCRIBE | is the DESCRIBE function |
| PURGE | is the PURGE function |
| QUIESCE | is the QUIESCE function |
| RECEIVE | is the RECEIVE function |
| REJECT | is the REJECT function |
| REPLY | is the REPLY function |
| RESUME | is the RESUME function |
| RTRVBFR | is the RETRIEVE BUFFER function |
| SEND | is the SEND function |

SETCMASK is the SET CONTROL MASK function
SETMASK is the SET MASK function
SEVER is the SEVER function
TESTCMPL is the TEST COMPLETION function
QUERY is the QUERY function
TESTMSG is the TEST MESSAGE function

ALL= (Used on QUIESCE, RESUME, SEVER)

ALL=YES specifies that the function requested is to be applied to all paths for this virtual machine.

The valid values for ALL= are YES and NO. If not specified, the default is NO.

If ALL=YES is specified, PATHID= is not allowed.

ANSBUF= (Used on REPLY, SEND)

This parameter specifies the area to contain the reply text of the message.

Specify either the relocatable label of the buffer or the number of a register that contains the address of the buffer. IUCV stores the address of the buffer in the function parameter list.

For SEND, this parameter identifies the area into which IUCV places the reply text.

For REPLY, this parameter identifies the area from which IUCV takes the reply text.

ANSBUF= is not valid on SEND if TYPE=1WAY is specified.

If this parameter is not specified, the macro assumes that either the parameter is not needed (such as a SEND with TYPE=1WAY) or the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

ANSLEN= (Used on REPLY, SEND)

This parameter specifies the length of the area specified on the ANSBUF parameter.

Specify either (1) the relocatable label of the location containing the buffer length, or (2) the number of a register that contains the length of the buffer. The macro assumes a halfword value for the length at the storage location specified, or in the low-order halfword of the register specified. If a length modifier of 4 is used, the macro uses the fullword value for the length at the storage location or in the register specified. IUCV stores the buffer length in the function parameter list. If this parameter is not specified, the macro assumes that either the parameter is not needed (such as on a SEND with TYPE=1WAY) or the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

ANSLEN= may be specified even though ANSBUF= is not. If ANSBUF= has not been specified, the macro assumes that the invoker has moved the address of the answer buffer into the parameter list prior to invoking the IUCV macro.

ANSLEN= is not valid on SEND if TYPE=1WAY is specified.

BUFFER= (Used on DCLBFR, RECEIVE, SEND)

When you invoke DCLBFR, this parameter identifies the external interrupt buffer. When an external interrupt is reflected to the virtual machine, IUCV stores information concerning the IUCV message or a control interrupt in this buffer.

When you invoke SEND, this parameter identifies the area from which IUCV takes the message text.

When you invoke RECEIVE, this parameter identifies the area into which IUCV places the message text.

Specify either the relocatable label of the buffer or the number of a register that contains the address of the buffer. IUCV stores the address of the buffer in the function parameter list.

If this parameter is not specified, the macro assumes that the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

BUFLEN= (Used on RECEIVE, SEND)

This parameter specifies the length of the area specified on the BUFFER= parameter.

Specify either (1) the relocatable label of the location containing the buffer length, or (2) the number of a register that contains the length of the buffer. The macro assumes a halfword value for the length at the storage location specified, or in the low-order halfword of the register specified. If a length modifier of 4 is used, the macro uses the fullword value for the length at the storage location or in the register specified. IUCV stores the buffer length in the function parameter list.

If this parameter is not specified, the macro assumes that the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

BUFLEN= may be specified even though BUFFER= is not. If BUFFER= has not been specified, the macro assumes that the invoker has moved the address of the buffer into the parameter list prior to invoking IUCV macro.

Do not use BUFLEN= for the DECLARE BUFFER function. By default, the buffer declared on the DECLARE BUFFER is 40 bytes long.

CP= (Used on ACCEPT, CONNECT, DESCRIBE, PURGE, QUIESCE, RECEIVE REJECT, REPLY, RESUME, RTRVBFR, SEND, SEVER, TESTCMPL)

Specify CP=NO when invoking an IUCV function from a virtual machine. IUCV generates the IUCV instruction. The code generated when you specify CP=NO modifies general register zero. The virtual machine must be in supervisor state when the IUCV macro executes.

CP=YES specifies that the function is being invoked from the CP system code. A CALL linkage to CP module DMKIUACP is generated instead of the IUCV instruction. The macro modifies general registers 0, 1, and 15. The invoker is responsible for providing an EXTRN statement for module DMKIUACP. General register 11 is assumed to contain the address of the VMBLOK on whose behalf the specified function is to be performed. See the section "Communication Between CP and a Virtual Machine" for details on IUCV communications initiated from CP system code.

The valid values for CP= are YES and NO. If not specified, the default is NO.

CP=YES is required to invoke IUCV functions from CP system code.

If CP=YES is specified, MSGTAG= is not allowed.

The DESCRIBE and TEST COMPLETION functions cannot be used in CP outside of IUCV support.

DATA= (Used on SEND, REPLY)

This parameter specifies the location of your message data for this IUCV communication.

If you specify DATA=PRMMSG, your message or reply data is contained in the parameter list. You may use the PRMMSG= parameter to have the message or reply data moved into the parameter list. When DATA= is specified, the IUCV macro parameters BUFFER and BUFLEN may not be used on the SEND function and the parameters ANSBUF and ANSLEN may not be used on the REPLY function.

If you specify DATA=BUFFER, your messages or reply is contained in a buffer. The IUCV macro parameter of PRMMSG= may not be used when DATA=BUFFER.

The DATA= option on SEND and REPLY are independent of each other. The protocol used is at the discretion of the communicators. You may define a protocol such that:

- A message specified in the parameter list using the DATA= option is REPLYed to via a message in the answer buffer specified on the SEND.

- A message sent in a buffer may be REPLYed to by the target via a message in the parameter list using the DATA= option.

FCNCD= (Used on CONNECT)

This parameter indicates which CP system service is invoking the CONNECT function. Each supported CP service is identified by a one-byte numerical code. VM/SP presently supports IUCV communication for only one CP Service, SNA Console Communication Services. CCS has a code of 0.

Specify either the code itself or the number of a register that contains the code in its low-order byte. IUCV moves the code into the function parameter list.

If this parameter is not specified, the macro assumes that the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

This parameter is valid only if CP=YES is specified.

MASK= (Used on SETMASK and SETCMASK)

This parameter specifies the mask byte to determine which, if any, of the IUCV external interrupts a virtual machine is to be enabled for. Specify either the relocatable label of a byte containing the mask, or the number of a register that contains the mask in its low-order byte. IUCV moves the mask into the function parameter list.

If this parameter is not specified, the macro assumes that the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

The MASK= parameter is valid only if you have specified CP=NO. The SET MASK and SET CONTROL MASK functions cannot be invoked from CP system code.

MF= (Used on ACCEPT, CONNECT, DCLBFR, DESCRIBE, PURGE, QUESCE, RECEIVE, REJECT, REPLY, RESUME, SEND, SETMASK, SETCMASK, SEVER, TESTCMPL)

The MF=L option is allowed as a keyword parameter on any IUCV function that uses a parameter list.

This paramater lets you initialize an IUCV parameter list without issuing the IUCV instruction (from a virtual machine) or the SVC (from CP system code). This parameter allows programs to initialize an IUCV parameter list and to pass that parameter list to an operating system which provides an IUCV interface (for example, CMS).

MSGID= (Used on PURGE, RECEIVE, REJECT, REPLY, TESTCMPL)

This parameter specifies the message identifier of the message to search for. The message identifier uniquely identifies a particular message. IUCV generates the message id and returns it in the SEND parameter list when a message is created.

Specify either the relocatable label of a fullword containing the message identifier, or the number of a register that contains the message identifier. IUCV stores the message identifier in the function parameter list.

If this parameter is not specified, the IUCV macro assumes that either the parameter is not needed (for example, when you specify a message by path id only), or the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

MSGID= is an optional input to the functions listed above. When a MSGID is specified, you must also supply the path id, and message class (SRCCLS for PURGE and TESTCMPL, TRGCLS for RECEIVE, REJECT and REPLY).

If you specify the MSGID= parameter on the IUCV macro, the IPFGMID flag in IPFLAGS1 is set when you invoke the PURGE, RECEIVE, REJECT or TEST COMPLETION functions.

MSGLIM= (Used on ACCEPT, CONNECT)

This parameter specifies the limit of outstanding messages to be allowed on the path established by this CONNECT. A message limit can also be specified on the IUCV directory control statement. If a message limit has been specified in the directory, the value you specify with this parameter of the IUCV macro must not exceed that limit.

Specify either the relocatable label of a halfword containing the message limit, or the number of a register that contains the message limit in the low-order halfword. IUCV stores the message limit in the function parameter list.

If this parameter is not specified, the macro assumes that either the parameter is not needed (the value from the directory or the default is to be used) or the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

If the message limit is not specified on the IUCV macro or directory control statement, or if the value has not been stored is the function parameter list, ten is the default message limit.

The maximum value that can be specified for the message limit is 255. For CP system code, (CP=YES specified), there is no overriding directory value. If MSGLIM is not specified, a default of 10 is assumed by IUCV.

The originator of the connection sets up the message limit for the path.

MSGTAG= (Used on SEND)

This parameter specifies the tag of the message created by invoking the SEND function. IUCV returns the message tag when the message completes.

Specify either a relocatable label for a fullword containing the tag or the number of a register that contains the tag. IUCV stores the tag in the function parameter list.

If you specify CP=YES, MSGTAG= is not needed. CP system code uses the MSGTAG field in the parameter list for internal linkage.

If this parameter is not specified, the macro assumes that either it is not valid (for example, if CP=YES is specified) or that the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

PATHID= (Used on ACCEPT, PURGE, QUIESCE, RECEIVE, REJECT, REPLY, RESUME, SEND, SEVER, TESTCMPL)

This parameter specifies the path identification associated with a message. IUCV assigns a path identification and returns the value in the CONNECT parameter list.

All further communications on a path must specify the path id that was returned from CONNECT. Path ids are sequential from X'0000' to the maximum number of connections allowed for this virtual machine. As paths are severed, the IUCV reuses vacated path ids.

Specify either the relocatable label of a halfword that contains the path id or the number of a register that contains the path id in the low-order halfword. IUCV stores the path identifier in the IUCV parameter list.

If this parameter is not specified, the macro assumes that either the parameter is not needed (for example, if you invoke the SEVER function with ALL=YES) or the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

If you specify MSGID on the PURGE, RECEIVE, REJECT, REPLY, or TEST COMPLETION functions, IUCV requires that you specify path id and message class (SRCCLS or TRGCLS, as appropriate).

PATHID= is not valid if ALL=YES is also specified.

If you specify the PATHID= parameter on the IUCV macro, the IPFGPID flag in IPFLAGS1 is set for PURGE, RECEIVE, REJECT and TEST COMPLETION functions.

PRMDATA= (Used on ACCEPT, CONNECT)

This parameter specifies whether the communicator wishes to allow messages that contain the message data in the parameter list (for example, messages sent via the DATA=PRMMSG option).

Specify PRMDATA=YES if you are willing to receive messages via the DATA=PRMMSG option in your parameter list.

Specify PRMDATA=NO if you are not willing to receive messages sent into your parameter list and only accept messages sent using a buffer. NO is the default if the PRMDATA parameter is not used.

PRMLIST= (Used on ACCEPT, CONNECT, DCLBFR, DESCRIBE PURGE, QUIESCE, RECEIVE, REJECT, REPLY, RESUME, SEND, SETMASK, SEVER, TESTCMPL)

This parameter identifies the IUCV parameter list, which is input to the actual IUCV instruction or CALL to DMKIUACP. This parameter list must be a real address if CP=YES (invoked from CP system code) or a guest real address (real to the virtual machine) if invoked from a virtual machine. The parameter list must be on a doubleword boundary.

Specify either a relocatable label or the number of a register. If a label is specified, the macro assumes it is the label of the parameter list. The address of the parameter list is loaded into general register 1 if CP=YES, or the IUCV instruction is generated to reference the label if CP=NO. If a register is specified, the macro assumes it contains the address of the parameter list; the address is loaded into general register 1 if CP= YES, or the IUCV instruction is generated to reference the register if CP=NO.

This parameter is required for all IUCV functions except QUERY, and TEST MESSAGE.

If CP system code issues a SEND or CONNECT, the area specified on this parameter must be the address of an IXBLOK instead of a parameter list. See the section, "Invoking Communications between CP and a Virtual Machine" for details.

PRMMSG= (Used on SEND, REPLY)

This parameter specifies the eight bytes of message data that are moved into the parameter list.

Specify either the relocatable label of the eight bytes of message data or the number of a register that contains the address of the data.

PRTY= (Used on ACCEPT, CONNECT, REPLY, SEND)

When you invoke the CONNECT function, PRTY=YES indicates that you want to establish a path that can handle priority communi-

cations. When invoked from a virtual machine, PRIORITY must be authorized in the IUCV directory entry. When invoked from CP system code (CP=YES), PRTY=YES is always valid.

When you invoke the SEND and REPLY functions, PFTY=YES indicates that this message or reply is a priority message. PRTY=YES is only valid if this path can handle priority communications.

Valid values for PRTY= are YES and NO. If not specified, the default is NO.

The originator of the connection sets up the priority for both ends of the path.

QUIESCE= (Used on ACCEPT, CONNECT)

QUIESCE=YES indicates that you want to quiesce the path being established; the other communicator cannot send messages on a quiesced path.

The valid values for QUIESCE= are YES and NO. If not specified, the default is NO.

You can restore the path to full communication capability by invoking the RESUME function.

SRCCLS= (Used on PURGE, SEND, TESTCMPL)

This parameter specifies the source message class associated with a message.

When you invoke the PURGE function, this parameter optionally specifies the source message class of the message to be purged. If omitted, IUCV does not use the source message class in the search for the message.

When you invoke the SEND function, this parameter specifies the source message class that IUCV stores in the MSGBLOK that represents the message.

When you invoke the TEST COMPLETION function, this parameter optionally specifies the source message class of the message to be dequeued. If omitted, IUCV dequeues the first message encountered on the specified path regardless of its source message class.

Specify either the relocatable label of a fullword containing the source message class or the number of a register containing the source message class. IUCV stores the source message class in the function parameter list.

If this parameter is not specified, the macro assumes either that the parameter is not needed (for example, if you invoke a PURGE by path id alone), or that the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

If you specify the SRCCLS= parameter on the IUCV macro for the PURGE and TEST COMPLETION functions, the IPFGMCL flag in IPFLAGS1 is set.

TRGCLS= (Used on RECEIVE, REJECT, REPLY, SEND)

This parameter specifies the target message class associated with this message.

When you invoke the RECEIVE and REJECT functions, this parameter optionally specifies the target message class of the message to be received for rejected. If omitted, IUCV does not use the target message class in the search for the message.

When you invoke the SEND function, this parameter specifies the target message class that IUCV stores in the MSGBLOK that represents the message.

When you invoke the REPLY function, this parameter specifies the target message class of the message being responded to.

Specify either the relocatable label of a fullword containing the target message class, or the number of a register containing the target message class. IUCV stores the target message class in the function parameter list.

If this parameter is not specified, the macro assumes that either the parameter is not needed (for example if you issue a RECEIVE by path id alone) or the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

If you specify the TRGCLS= parameter on the IUCV macro for the RECEIVE and REJECT functions, the IPFGMCL flag in IPFLAGS1 is set.

TYPE= (Used on SEND)

TYPE=1WAY specifies that this is a one-way transaction. No REPLY by the receiver is needed or valid. IUCV moves the MSGBLOK representing the message to the source communicator's REPLY queue when the target communicator issues a RECEIVE for the message. TYPE=2WAY specifies that this is a two-way transaction. IUCV moves the message to the source's REPLY queue only when the target invokes a REPLY for this message. Two way transactions are useful for returning data in response to a specific request.

The valid values for TYPE= are 1WAY and 2WAY. If not specified, the default is 2WAY.

USERDTA= (Used on ACCEPT, CONNECT, QUIESCE, RESUME, SEVER)

This parameter specifies the 16-byte user data area that is to be reflected to the target.

Specify either (1) the relocatable label of the storage area, or (2) the number of a register that contains the address of the user data storage area. IUCV moves the address of the storage area into the function parameter list.

If this parameter is not specified, the macro assumes that the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

USERID= (Used on CONNECT)

This parameter specifies the eight-character userid of the virtual machine or CP system service to which you want to establish this path.

Specify either the relocatable label of the storage area containing the userid, or the number of a register that contains the address of the userid. IUCV stores the userid in the function parameter list.

If this parameter is not specified, the IUCV macro assumes that the invoker has stored a value in the parameter list prior to invoking the IUCV macro.

VMBLOK= (Used on ACCEPT, CONNECT, DESCRIBE, PURGE, QUIESCE, RECEIVE, REJECT, REPLY, RESUME, RTRVBFR, SEND, SEVER, TESTCMPL)

VMBLOK=USER specifies that the IUCV control blocks associated with the current VMBLOK are to be used for this IUCV request.

VMBLOK=SYSTEM specifies that the IUCV control blocks associated with the system VMBLOK are to be used for this IUCV request.

The valid values for VMBLOK= are USER and SYSTEM. If not specified, the default is SYSTEM.

VMBLOK= is only valid if CP=YES is specified.

See Figure 18 on page 139 for a reference to the relationships between the IUCV functions the IUCV macro instruction keyword parameters.

| IUCV Macro Parameters | ACCEPT | CONNECT | DCLBFR | DESCRIBE | PURGE | QUIESCE | RECEIVE | REJECT | REPLY | RESUME | RTRVBFR | SEND | SETCMASK | SETMASK | SEVER | TESTCMPL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ALL | | | | | | X | | | | X | | | | | X | |
| ANSBUF | | | | | | | | | X | | | X | | | | |
| ANSLEN | | | | | | | | | X | | | X | | | | |
| BUFFER | | | X | | | | X | | | | | X | | | | |
| BUFLEN | | | | | | | X | | | | | X | | | | |
| CP | X | X | | X | X | X | X | X | X | X | X | X | | | X | X |
| DATA | | | | | | | | | X | | | X | | | | |
| FCNCD | | X | | | | | | | | | | | | | | |
| MASK | | | | | | | | | | | | | X | X | | |
| MF | X | X | X | X | X | X | X | X | X | X | | X | X | X | X | X |
| MSGID | | | | | X | | X | X | X | | | | | | | X |
| MSGLIM | X | X | | | | | | | | | | | | | | |
| MSGTAG | | | | | | | | | | | | X | | | | |
| PATHID | X | | | | X | X | X | X | X | X | | X | | | X | X |
| PRMDATA | X | X | | | | | | | | | | | | | | |
| PRMLIST | X | X | X | X | X | X | X | X | X | X | | X | X | X | X | X |
| PRMMSG | | | | | | | | | X | | | X | | | | |
| PRTY | X | X | | | | | | | X | | | X | | | | |
| QUIESCE | X | X | | | | | | | | | | | | | | |
| SRCCLS | | | | | X | | | | | | | X | | | | X |
| TRGCLS | | | | | | | X | X | X | | | X | | | | |
| TYPE | | | | | | | | | | | | X | | | | |
| USERDTA | X | X | | | | X | | | | X | | | | | X | |
| USERID | | X | | | | | | | | | | | | | | |
| VMBLOK | X | X | | X | X | X | X | X | X | X | X | X | | | X | X |

**Notes:**

1. PRMLIST is a required parameter (others are optional).
2. The QUERY and TEST MESSAGE functions do not use parameters.

**Figure 18. IUCV Function and IUCV Macro Parameter Relationships**

### *Invoking Communications between CP and a Virtual Machine*

Specify CP=NO when invoking an IUCV function from a virtual machine. The IUCV instruction is generated. If a label is specified for the parameter list, it must be relocatable and addressable. The code generated by CP=NO modifies general register 0. When the function is executed, the virtual machine must be in supervisor state. CP=NO is the default.

CP system services invoke the IUCV macro instruction specifying CP=YES. CP=YES generates a CALL linkage directly to the IUCV processing module (DMKIUACP). If a label is specified for the parameter list, it must be relocatable and addressable. The code generated by CP=YES modifies general registers 0, 1 and 15. The invoker must supply an EXTRN statement for the entry point DMKIUACP.

If VMBLOK=USER is specified with CP=YES, then a CALL linkage is generated directly to the IUCV processing module (DMKIUACU). The invoker must supply an EXTRN statement for the entry point DMKIUACU.

### *Requests Initiated by the Virtual Machine*

When a virtual machine wishes to establish communications with a CP system service, it invokes the CONNECT function specifying the name of the desired CP service as the target virtual machine ID.

The IUCV communication processor receives control from the CONNECT function, gathers the external interrupt information and determines which service is desired. The communication processor then locates the CONNECT entry point for that service and, using CALL linkage, passes control to that entry point.

The CONNECT entry point for the requested CP system service inspects the external interrupt data. It must either accept the connection or reject the connection. To accept the connection, it invokes the ACCEPT function, specifying CP=YES. To reject the connection, it invokes the SEVER function specifying CP=YES. When the service module has finished responding to the incoming connection request, it issues an EXIT (SVC 12) to return control to the communications processor.

When an incoming message for a CP system service is encountered, the communications processor gathers the external interrupt information and determines which service is desired. The communication processor locates the entry point that processes incoming messages for the desired service and, using CALL linkage, passes control to it.

The message processing module of the CP service then inspects the external interrupt data. The CP service module must invoke the RECEIVE function, specifying CP=YES, to obtain the actual message. When the RECEIVE function completes, the message data will have been moved to the address specified in the RECEIVE parameter list. The CP service module then interprets the message data and services the request. When the request has been satisfied, the CP service module invokes the REPLY function to satisfy the two-way message protocol. When the REPLY function completes, the reply has been queued back to the source communicator. When the CP service module completes processing of the message, it issues an EXIT (SVC 12) to return to the communications processor.

When a virtual machine wishes to terminate a communications path, it invokes the SEVER function via the IUCV macro. The communication processor receives control from the SEVER function, gathers the external interrupt information, and determines which service was connected. The communication processor locates the SEVER entry point for that service and, using CALL linkage, passes control to it.

The SEVER entry point for that CP system service then inspects the external interrupt data. The CP system service module issues a SEVER if the connection was complete. When the CP service module finishes processing, it issues an EXIT (SVC 12) to return control to the communication processor.

If a virtual machine wishes to quiesce a communications path, it invokes the QUIESCE function of the IUCV macro. The communications processor receives control from the QUIESCE function, gathers the external interrupt information, and determines which service was connected. The communications processor locates the QUIESCE entry point for that service and, using CALL linkage, passes control to it.

The QUIESCE entry point for the CP system service then inspects the external interrupt data. The CP service records the fact that the path has been quiesced. When the CP service module has finished processing, it issues an EXIT (SVC 12) to return control to the communication processor.

After invoking QUIESCE for a path, the virtual machine may eventually invoke the RESUME function for the path.

The communication processor receives control from the RESUME function, gathers the external interrupt information, and determines which service was connected. The communication processor locates the RESUME entry point for that service and, using CALL linkage, passes control to it.

The RESUME entry point for that CP system service then inspects the external interrupt data. The CP service records the fact that the path has been RESUMEd. When the CP service module has finished processing, it issues an EXIT (SVC 12) to return control to the communication processor.

### *CP Initiated Requests*

When a CP module initiates a CONNECT or SEND to a virtual machine, it must do the following:

- Get storage (via DMKFREE) in which to build an IXBLOK.
- Build the parameter list in the IXBLOK for the function that it wishes to invoke.
- Store the general registers in the IXBLOK.
- Store the address of the routine that gets control when a connection completes or when a reply is received. The CP module must store the routine's address in the "interrupt return address" field of the IXBLOK (label IXIRA).
- Invoke the CONNECT or SEND function via the IUCV macro, specifying CP=YES and specifying the address of the IXBLOK or the PRMLIST=parameter.

When the function has been initiated, control returns to the next sequential instruction after the IUCV macro instruction. When the function completes (that is, when the target communicator invokes the ACCEPT or REPLY function), the communications processor gets control. The communications processor loads the general registers from the IXBLOK and passes control to the routine at the "interrupt return addresscdq.. The communications processor restores all registers except register 15 from the IXBLOK. Register 15 is used in passing control and is loaded with the interrupt return address.

The CP module that builds the IXBLOK is responsible for the following:

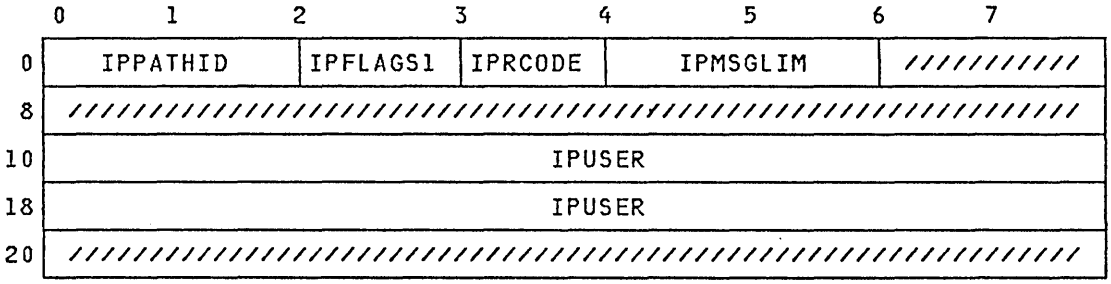
- Restoring the base register for the module that invoked the function (pass the base register in general register 12 following CP conventions).
- Releasing (via DMKFRET) the IXBLOK upon completion of the asynchronous portion of the function (pass the address of the IXBLOK in one of the general registers).
- Lock the processing module in real storage if it is not resident. In addition, when the routine at the interrupt return address gets control, the processing module must be unlocked.

## IUCV Parameter List Formats

This section illustrates the formats of the parameter lists required for IUCV functions. Descriptions of the parameter list fields are included in the section, "Parameter List and External Interrupt Fields."

### ACCEPT Function

ACCEPT Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPFLAGS1 IPMSGLIM IPPATHID IPUSER

- OUTPUTs from this function returned in the parameter list:

  IPMSGLIM IPRCODE

- Input flags for this function (set by the IUCV macro or by the invoker in IPFLAGS1):

  IPQUSCE — Connect in quiesce mode (the originator of the connection will be unable to issue SENDs).

  IPPRTY — The connection established can handle priority messages.

  IPRMDATA — The communicator is prepared to handle message data in his parameter list.

- Output flags for this function (returned by IUCV in IPFLAGS1):

  IPPRTY — Priority messages are allowed for this connection.

- Exceptions generated by this function (ABENDs generated for CP system code):

  Specification — Parameter list not on a doubleword boundary.

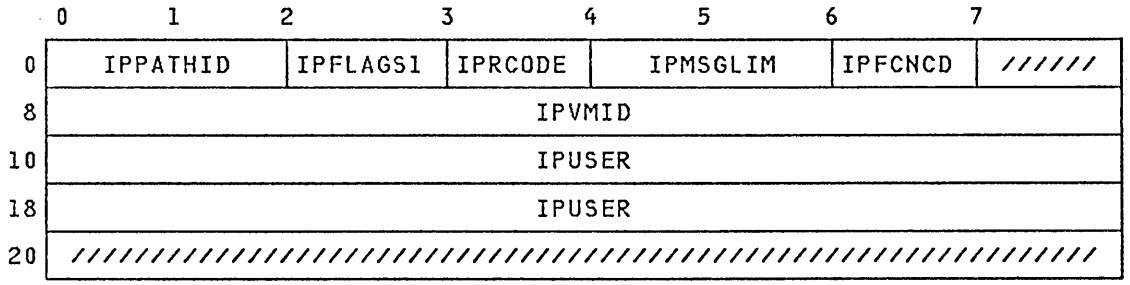
  Operation — An external interrupt buffer has not been declared via the DCLBFR function, or the invoker is not in supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared.

  Addressing — Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address.

  Protection — Invalid parameter list address. The storage key of the specified address does not match the key of the user.

**CONNECT Function**

CONNECT Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPFCNCD IPFLAGS1 IPMSGLIM IPUSER IPVMID

- OUTPUTs from this function returned in the parameter list:

  IPMSGLIM IPPATHID IPRCODE

- Input flags for this function (set by the IUCV macro or by the invoker in IPFLAGS1):

  IPPRTY — The connection established can handle priority messages.

  IPQUSCE — Connect in Quiesce mode. (The target communicator cannot issue SENDs).

  IPRMDATA — The communicator is prepared to handle message data in his parameter list.

- Output flags for this function (returned by IUCV in IPFLAGS1):

  IPPRTY — This is a priority message.

  IPPRTY — Priority messages are allowed for this connection.

- Exceptions generated by this function (ABENDs generated for CP system code):

  Specification — Parameter list not on a doubleword boundary.

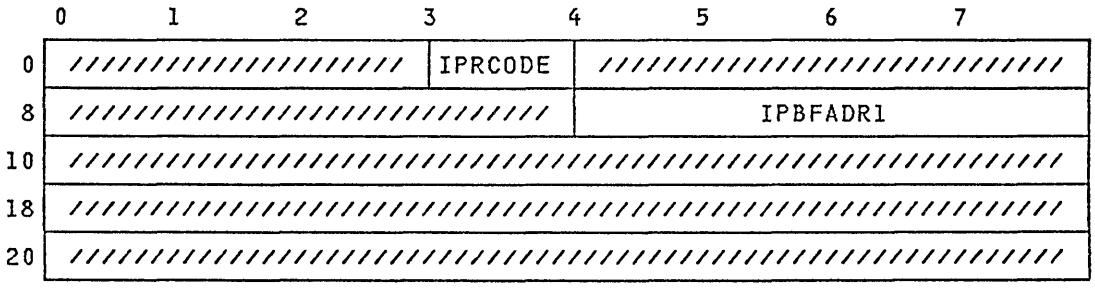
  Operation — An external interrupt buffer has not been declared via the DCLBFR function, or the invoker is not in supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared.

  Addressing — Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address.

  Protection — Invalid parameter list address. The storage key of the specified address does not match the key of the user.

**DECLARE BUFFER Function**

DCLBFR Parameter List Format



- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPBFADR1

- OUTPUTs from this function returned in the parameter list:

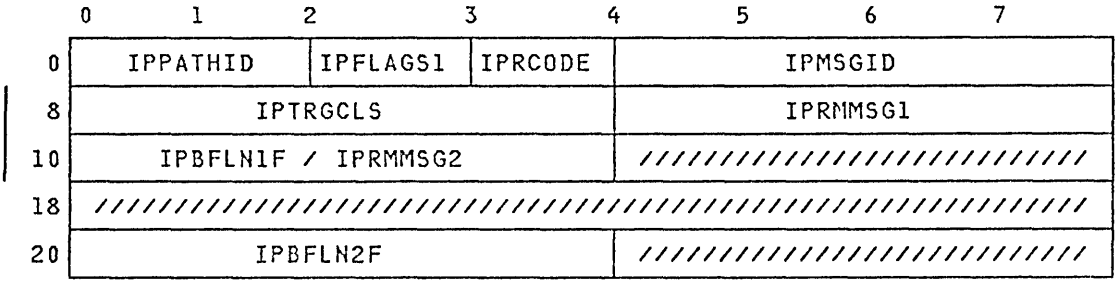
  IPRCODE

- Exceptions generated by this function (ABENDs generated for CP system code):

| | |
|---|---|
| Specification | Parameter list not on a doubleword boundary. |
| Addressing | Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address. |
| | Invalid buffer address. |
| Operation | Invoker not in supervisor state. |
| Protection | Invalid parameter list address. The storage key of the specified address does not match the key of the user. |

**DESCRIBE Function**

DESCRIBE Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  NONE

- OUTPUTs from this function returned in the parameter list:

  IPBFLN1 IPPATHID IPRCODE IPBFLN1F IPRMMSG1
  IPBFLN2 IPMSGID IPTRGCLS IPBFLN2F IPRMMSG2
  IPFLAGS1

- Output flags for this function (returned in IPFLAGS1):

IPFGMCL — Always returned as 1 so that the resulting parameter list is valid input to the next function (normally RECEIVE or REPLY).

IPFGMID — Always returned as 1 so that the resulting parameter list is valid input to the next function (normally RECEIVE or REPLY).

IPFGPID — Always returned as 1.

IPNORPY — This is a one-way type message.

IPPRTY — This is a priority message.

IPRMDATA — The message data is in the IPRMMSGx fields of the parameter list.

- Exceptions generated by this function (ABENDs generated for CP system code):

Specification — Parameter list not on a doubleword boundary.

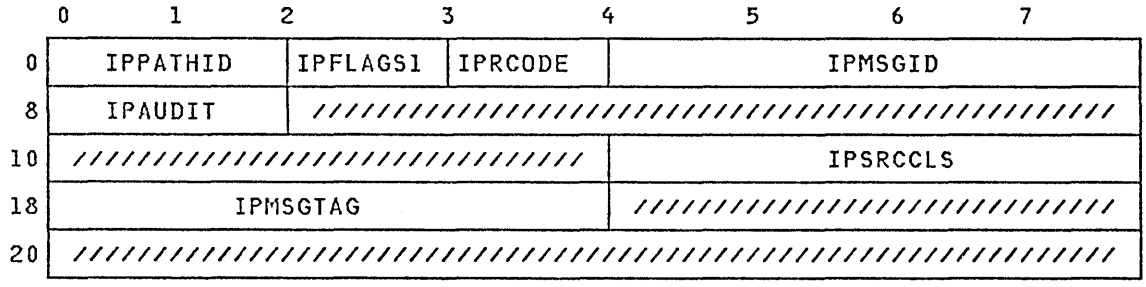
Operation — An external interrupt buffer has not been declared via the DCLBFR function, or the invoker is not in supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared.

Addressing — Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address.

Protection — Invalid parameter list address. The storage key of the specified address does not match the key of the user.

**PURGE Function**

PURGE Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

IPFLAGS1 IPMSGID IPPATHID IPSRCCLS

- OUTPUTs from this function returned in the parameter list:

  IPAUDIT IPMSGID IPPATHID IPRCODE IPSRCCLS
  IPFLAGS1 IPMSGTAG

- Input flags for this function (set by the IUCV macro or by the invoker in IPFLAGS1):

  | | |
  |---|---|
  | IPFGMCL | A message class identifier (SRCCLS) has been supplied in the parameter list. |
  | IPFGMID | A message identifier has been supplied in the parameter list. |
  | IPFGPID | A path identifier has been supplied in the parameter list. |

- Output flags for this function (returned by IUCV in IPFLAGS1):

  | | |
  |---|---|
  | IPNORPY | This is a one-way type message. |
  | IPPRTY | This is a priority message. |

- Exceptions generated by this function (ABENDs generated for CP system code):

  | | |
  |---|---|
  | Specification | Parameter list not on a doubleword boundary.<br><br>Invalid search flags. Either the path id has not been specified, or the message id has been specified without a message class. |
  | Operation | An external interrupt buffer has not been declared via the DCLBFR function, or the invoker is not in supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared. |
  | Addressing | Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address. |
  | Protection | Invalid parameter list address. The storage key of the specified address does not match the key of the user. |

**QUERY Function**

The QUERY function does not take a parameter list.

The QUERY function is used to obtain IUCV information about a virtual machine.

QUERY RESULTS:

The size of the IUCV external interrupt buffer is returned in general register 0.

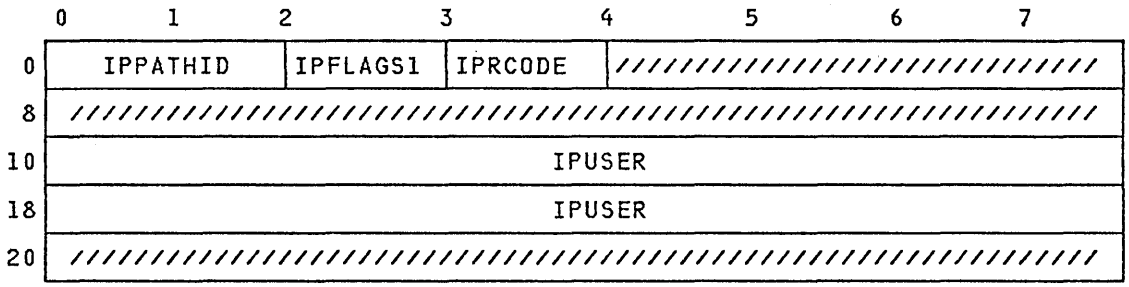
The maximum number of connections that can be outstanding for this virtual machine is returned in general register 1.

- Exceptions generated by this function:

  Operation — Invoker not in supervisor state.

### QUIESCE Function

QUIESCE Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPFLAGS1 IPPATHID IPUSER

- OUTPUTs from this function returned in the parameter list:

  IPRCODE

- Input flags for this function (set by the IUCV macro or by the invoker in IPFLAGS1):

  IPALL — Quiesce all paths for this virtual machine.

- Exceptions generated by this function (ABENDs generated for CP system code):

  Specification — Parameter list not on a doubleword boundary.

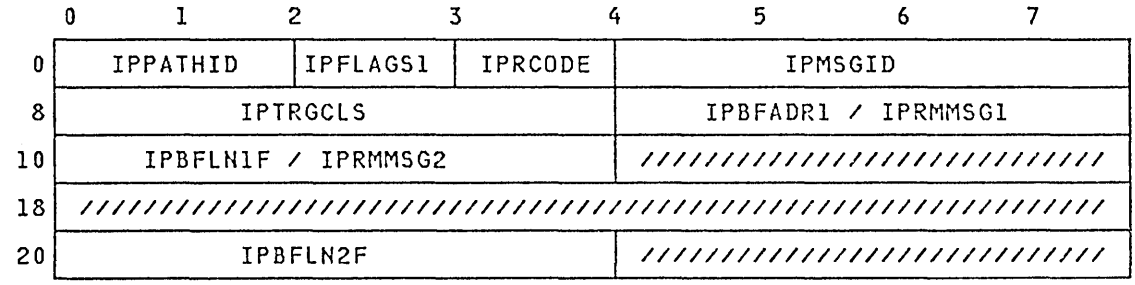
  Operation — An external interrupt buffer has not been declared via the DCLBFR function, or the invoker is not in supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared.

  Addressing — Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address.

| Protection | Invalid parameter list address. The storage key of the specified address does not match the key of the user. |
|---|---|

**RECEIVE Function**

RECEIVE Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPBFADR1 IPFLAGS1 IPMSGID IPPATHID IPTRGCLS
  IPBFLN1 IPBFLN1F IPBFLN2F

- OUTPUTs from this function returned in the parameter list:

  IPBFLN1 IPBFADR1 IPMSGID IPRCODE IPTRGCLS
  IPBFLN2 IPFLAGS1 IPPATHID IPRMMSG1 IPRMMSG2

- Input flags for this function (set by the IUCV macro or by the invoker in IPFLAGS1):

  | | |
  |---|---|
  | IPFGMCL | A message class identifier (TRGCLS) has been supplied in the parameter list. |
  | IPFGMID | A message id has been supplied in the parameter list. |
  | IPFGPID | A path id has been supplied in the parameter list. |

- Output flags for this function (returned in IPFLAGS1):

  | | |
  |---|---|
  | IPNORPY | This is a one-way type message. |
  | IPPRTY | This is a priority message |
  | IPRMDATA | The message data is in the IPRMMSGx fields of the parameter list. |

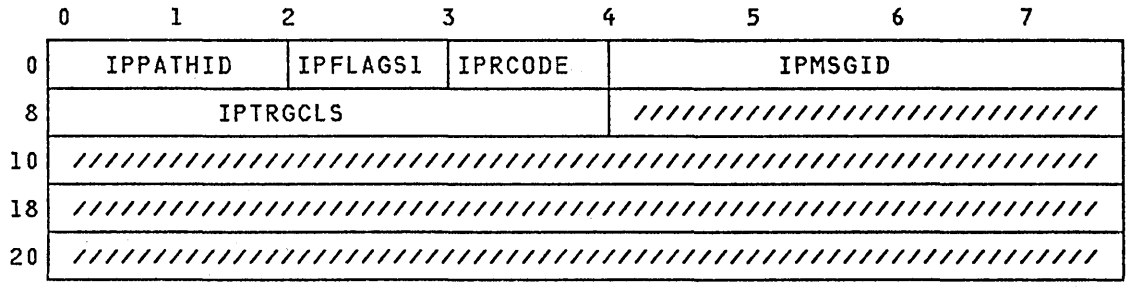
- Exceptions generated by this function (ABENDs generated for CP system code):

  | | |
  |---|---|
  | Specification | Parameter list not on a doubleword boundary. |

| | |
|---|---|
| | Invalid search flags. Message id has been specified without path id and message class. |
| Operation | An external interrupt buffer has not been declared via the DCLBFR function, or the invoker is not in supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared. |
| Addressing | Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address. |
| | Invalid buffer address. |
| Protection | Invalid parameter list address. The storage key of the specified address does not match the key of the user. |

**REJECT Function**

REJECT Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPFLAGS1 IPMSGID IPPATHID IPTRGCLS

- OUTPUTs from this function returned in the parameter list:

  IPMSGID IPPATHID IPRCODE IPTRGCLS

- Input flags for this function (set by the IUCV macro or by the invoker in IPFLAGS1):

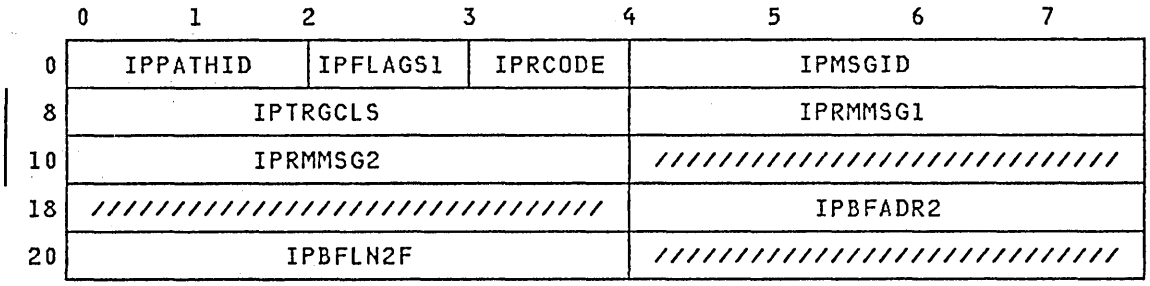
  | | |
  |---|---|
  | IPFGMCL | A message class id has been supplied in the parameter list. |
  | IPFGMID | A message id has been supplied in the parameter list. |
  | IPFGPID | A path id has been supplied in the parameter list. |

- Exceptions generated by this function (ABENDs generated for CP system code):

| | |
|---|---|
| Specification | Parameter list not on a doubleword boundary. |
| | Invalid search flags. Message id has been specified without path id and message class. |
| Operation | An external interrupt buffer has not been declared via the DCLBFR function, or the invoker is not in supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared. |
| Addressing | Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address. |
| Protection | Invalid parameter list address. The storage key of the specified address does not match the key of the user. |

**REPLY Function**

REPLY Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPBFADR2 IPBFLN2 IPFLAGS1 IPMSGID IPPATHID
  IPTRGCLS IPBFLN2F IPRMMSG1 IPRMMSG2

- OUTPUTs from this function returned in the parameter list:

  IPRCODE

- Input flags for this function (set by the IUCV macro or by the invoker in IPFLAGS1):

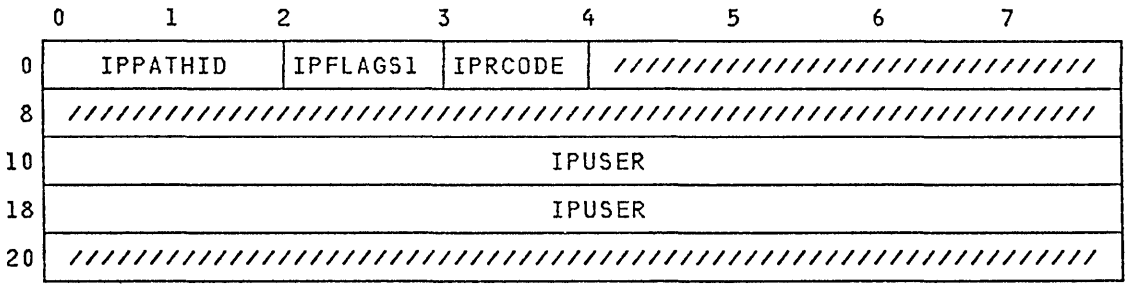
  | | |
  |---|---|
  | IPPRTY | This is a priority reply. |
  | IPRMDATA | The message data is in the IPRMMSGx fields of the parameter list. |

- Exceptions generated by this function (ABENDs generated for CP system code):

| | |
|---|---|
| Specification | Parameter list not on a doubleword boundary. |
| Operation | An external interrupt buffer has not been declared via the DCLBFR function, or the invoker is not in supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared. |
| Addressing | Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address. |
| | Invalid buffer address. |
| Protection | Invalid parameter list address. The storage key of the specified address does not match the key of the user. |

**RESUME Function**

RESUME Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPFLAGS1 IPPATHID IPUSER

- OUTPUTs from this function returned in the parameter list:

  IPRCODE

- Input flags for this function (set by the IUCV macro or by the invoker in IPFLAGS1):

  | | |
  |---|---|
  | IPALL | Resume all paths for this virtual machine. |

- Exceptions generated by this function (ABENDs generated for CP system code):

  | | |
  |---|---|
  | Specification | Parameter list not on a doubleword boundary. |
  | Operation | An external interrupt buffer has not been declared via the DECLARE BUFFER function, or the invoker is not in |

supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared.

Addressing — Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address.

Protection — Invalid parameter list address. The storage key of the specified address does not match the key of the user.

**RETRIEVE BUFFER Function**

The Retrieve Buffer function does not take a parameter list.

- Exceptions generated by this function (ABENDs generated for CP system code):

  Operation — An external interrupt buffer has not been declared via the DECLARE BUFFER function.

  Invoker not in supervisor state.

**SEND Function**

SEND Parameter List Format

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | IPPATHID | | IPFLAGS1 | IPRCODE | IPMSGID | | | |
| 8 | IPTRGCLS | | | | IPBFADR1 / IPRMMSG1 | | | |
| 10 | IPBFLN1F / IPRMMSG2 | | | | IPSRCCLS | | | |
| 18 | IPMSGTAG | | | | IPBFADR2 | | | |
| 20 | IPBFLN2F | | | | //////////////////////////// | | | |

- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPBFADR1 IPBFLN1 IPPATHID IPBFLN1F IPRMMSG1
  IPBFADR2 IPBFLN2 IPMSGTAG IPBFLN2F IPRMMSG2
  IPTRGCLS IPFLAGS1 IPSRCCLS

- OUTPUTs from this function returned in the parameter list:

  IPMSGID IPRCODE

- Input flags for this function (set by the IUCV macro or by the invoker in IPFLAGS1):

  IPNORPY — This is a one-way type message.

  IPPRTY — This is a priority message.

  IPRMDATA — The message data is in the IPRMMSGx fields of the parameter list.

- Exceptions generated by this function (ABENDs generated for CP system code):

  Specification — Parameter list not on a doubleword boundary.

  Operation — An external interrupt buffer has not been declared via the DECLARE BUFFER function, or the invoker is not in supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared.

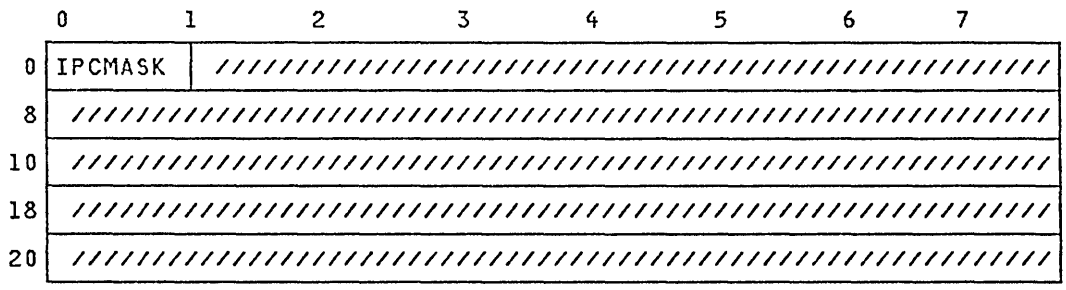
  Addressing — Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address.

  Invalid buffer address.

  Protection — Invalid parameter list address. The storage key of the specified address does not match the key of the user.

**SET CONTROL MASK Function**

SETCMASK Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPCMASK

- OUTPUTs from this function returned in the parameter list:

  NONE

- The bits defined in the IPCMASK field are:

  IPCLPC - X'80' - Enable for Type 01 - Pending connection

  IPCLCC - X'40' - Enable for Type 02 - Connection complete

  IPCLPS - X'20' - Enable for Type 03 - Path severed

  IPCLPQ - X'10' - Enable for Type 04 - Path quiesced

  IPCLPR - X'08' - Enable for Type 05 - Path resumed

  X'04' - Reserved (Should be set to zero)

  X'02' - Reserved (Should be set to zero)

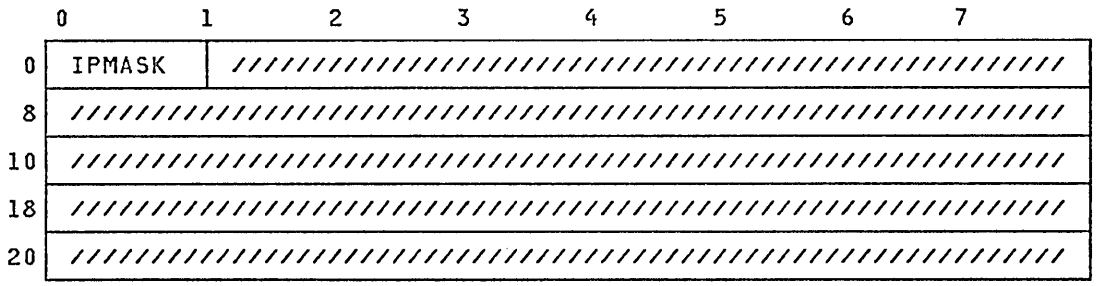
  X'01' - Reserved (Should be set to zero)

- Exceptions generated by this function (ABENDs generated for CP system code):

| | |
|---|---|
| Operation | An external interrupt buffer has not been declared via the DECLARE BUFFER function. |
| | Invoker not in supervisor state. |
| Protection | Invalid parameter list address. The storage key of the specified address does not match the key of the user. |

**SET MASK Function**

SETMASK Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPMASK

- OUTPUTs from this function returned in the parameter list:

  NONE

- Mask bits defined in the IPMASK field are:

  IPSNDN - X'80' - Enable for nonpriority message interrupts.

  IPSNDP - X'40' - Enable for priority message interrupts.

  IPRPYN - X'20' - Enable for nonpriority reply interrupts.

  IPRPYP - X'10' - Enable for priority reply interrupts.

  IPCTRL - X'08' - Enable for IUCV control interrupts.

  X'04' - RESERVED (Should be set to zero)

  X'02' - RESERVED (Should be set to zero)

  X'01' - RESERVED (Should be set to zero)

- Exceptions generated by this function (ABENDs generated for CP system code):

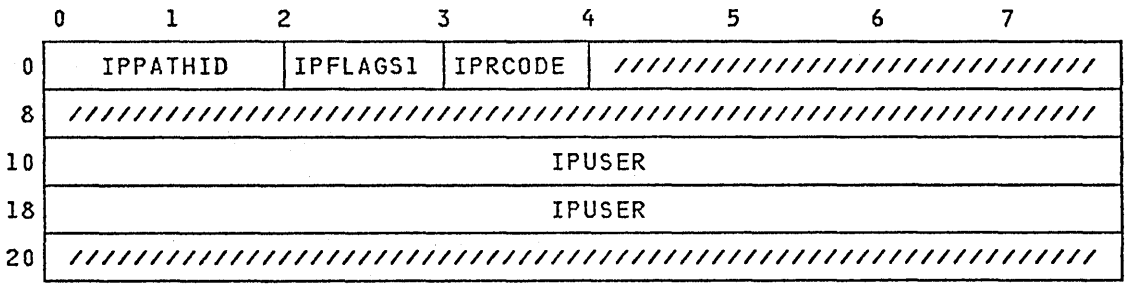
  Operation — An external interrupt buffer has not been declared via the DCLBFR function.

  Invoker not in supervisor state.

  Protection — Invalid parameter list address. The storage key of the specified address does not match the key of the user.

**SEVER Function**

SEVER Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPFLAGS1 IPPATHID IPUSER

- OUTPUTs from this function returned in the parameter list:

  IPRCODE

- Input flags for this function (set by the IUCV macro or by the invoker in IPFLAGS1):

  IPALL — Sever all paths for this virtual machine.

- Exceptions generated by this function (ABENDs generated for CP system code):

  Specification — Parameter list not on a doubleword boundary.

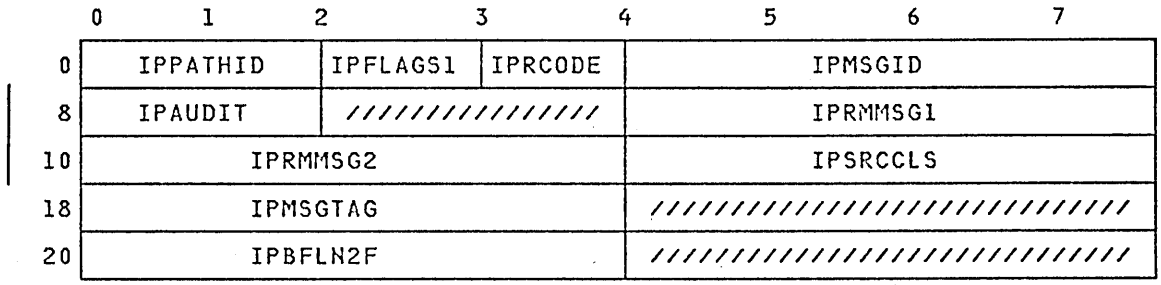
  Operation — An external interrupt buffer has not been declared via the DCLBFR function, or the invoker is not in supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared.

  Addressing — Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address.

  Protection — Invalid parameter list address. The storage key of the specified address does not match the key of the user.

**TEST COMPLETION Function**

TESTCMPL Parameter List Format




- INPUTs to this function (built in the parameter list by the IUCV macro or by the invoker):

  IPFLAGS1 IPMSGID IPPATHID IPSRCCLS

- OUTPUTs from this function returned in the parameter list:

  IPAUDIT IPFLAGS1 IPMSGTAG IPRCODE
  IPBFLN2 IPMSGID IPPATHID IPBFLN2F
  IPSRCCLS IPRMMSG1 IPRMMSG2

- Input flags for this function (set by the IUCV macro or by the invoker in IPFLAGS1):

  IPFGMCL — A message class has been supplied in the parameter list.

IPFGMID — A message id has been supplied in the parameter list.

IPFGPID — A path id has been supplied in the parameter list.

- Output flags for this function (returned in IPFLAGS1):

IPNORPY — This is a one-way message.

IPPRTY — This is a priority message.

IPRMDATA — The message data is in the IPRMMSGx fields of the parameter list.

- Exceptions generated by this function (ABENDs generated for CP system code):

Specification — Parameter list not on a doubleword boundary.

Invalid search flags. Message id has been specified without path id and message class.

Operation — An external interrupt buffer has not been declared via the DCLBFR function, or the invoker is not in supervisor state. When the function is invoked by CP system code, an operation exception cannot occur because an external interrupt buffer has not been declared.

Addressing — Invalid parameter list address. The specified address is outside the virtual machine or, for CP system code, is an invalid real address.

Protection — Invalid parameter list address. The storage key of the specified address does not match the key of the user.

### TEST MESSAGE Function

The TEST MESSAGE function does not use a parameter list.

- Exceptions generated by this function:

Operation — Buffer has not been declared via the DCLBFR function.

Invoker not in supervisor state.

## IUCV External Interrupt Formats

The following figures represent the content and format of the data presented on each of the IUCV external interrupts.

### External Interrupt for Pending Connection

When a virtual machine or CP system service invokes the CONNECT function, an external interrupt is reflected to the target virtual machine or passed by the IUCV communications processor to the CONNECT entry point of the requested CP system service.

The format and content of the external interrupt data is:

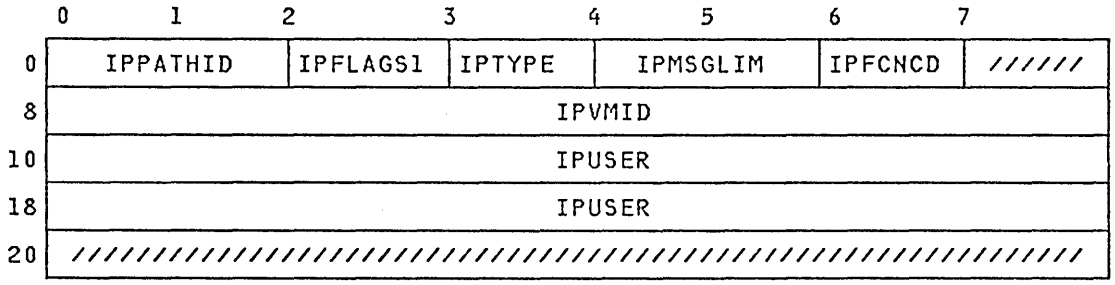



**Figure 19. Pending Connection External Interrupt Format**

### External Interrupt for Complete Connection

When CONNECT invoked by a virtual machine or CP system service has been responded to by the target virtual machine or CP system service, the external interrupt data has the following format:

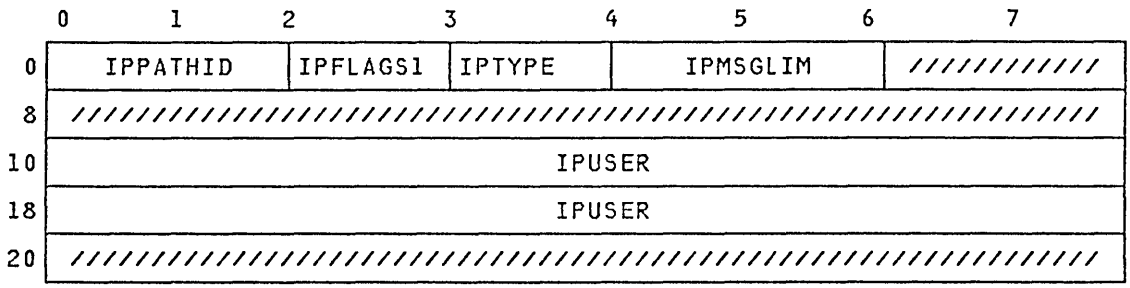



**Figure 20. Connection Complete External Interrupt Format**

### External Interrupt for Pending Messages

When a message is pending for a communicator, the external interrupt data has the following format. Note that the format of this data is the same as the output of the DESCRIBE function.

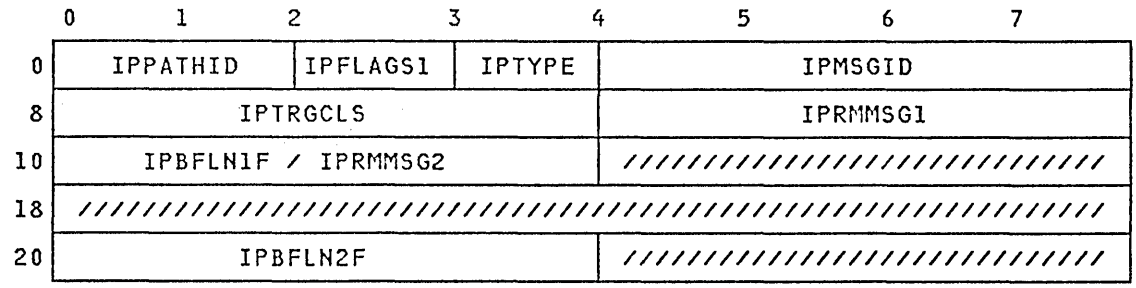



**Figure 21. Incoming Message External Interrupt Format**

### External Interrupt for Complete Messages

When a message is complete for a communicator, the external interrupt data has the following format. Note that the format of this data is the same as the output of the TEST COMPLETION function.

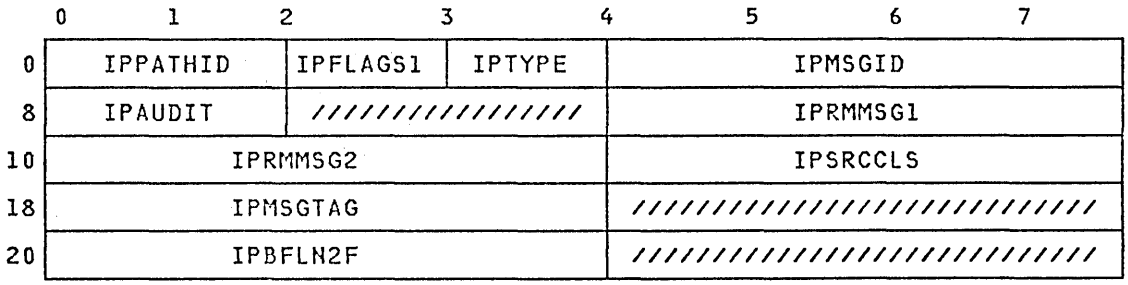



**Figure 22. Message Complete External Interrupt Format**

### External Interrupt from SEVER, QUIESCE, RESUME

When a SEVER, QUIESCE, or RESUME function has been invoked for a path to a virtual machine or CP system service, an external interrupt is generated. No action need be taken by a virtual machine or CP system service on a QUIESCE or RESUME. The interrupt is reflected so that the status of the path can be recorded.

When the SEVER function has been invoked by the communicating partner, this communicator must also invoke SEVER.

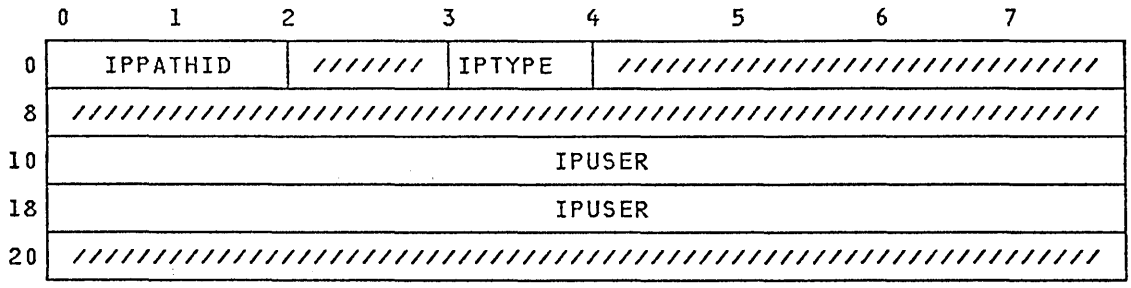



**Figure 23. SEVER, QUIESCE, RESUME External Interrupt Format**

### Parameter List and External Interrupt Fields

The following paragraphs define the fields of the IUCV parameter lists and external interrupts. Not every field has meaning for every function. This section explains, for each field, the functions for which this field is valid, and the meaning or use of the field.

IPAUDIT (Output from PURGE, TESTCMPL)

(Reflected in the message-complete IUCV external interrupt)

IUCV returns the audit trail of the message in this field. If no message was found, this field is not modified. One output of the TEST COMPLETION function, occurs when this field contains any nonzero bits. In this case, a condition code of 3 is set. This indicates that IUCV has stored a nonzero audit trail.

The meanings of the bits in the audit trail are:

| | | |
|---|---|---|
| IPADRPLE | X'8000' | Reply too long for buffer |
| IPADSNPX | X'4000' | Protection exception on send buffer |
| IPADSNAX | X'2000' | Addressing exception on send buffer |
| IPADANPX | X'1000' | Protection exception on answer buffer |
| IPADANAX | X'0800' | Addressing exception on answer buffer |
| IPADRJCT | X'0400' | Message was rejected |
| IPADPRMD | X'0200' | Reply data was specified using DATA=PRMMSG and originator did not allow this option. |
| | X'0100' | Reserved |
| IPADRCPX | X'0080' | Protection exception on receive buffer |
| IPADRCAX | X'0040' | Addressing exception on receive buffer |
| IPADRPPX | X'0020' | Protection exception on reply buffer |
| IPADRPAX | X'0010' | Addressing exception on reply buffer |
| IPADSVRD | X'0008' | Path was severed |
| | X'0004' | Reserved |
| | X'0002' | Reserved |
| | X'0001' | Reserved |

IPBFADR1 (Input to DCLBFR, RECEIVE, SEND)

(Output from RECEIVE)

This field specifies the address of the area to contain the text of the message or the area to be used for storing IUCV external interrupt information.

As input to SEND, this field identifies the area from which IUCV takes the message text.

As input to RECEIVE, this field identifies the area into which IUCV places the message text.

As input to DCLBFR, this field identifies the area into which IUCV stores information concerning an IUCV external interrupt.

The contents of this field are updated by the RECEIVE function. When the function is finished, the address has been increased by the length of the data received.

IPBFADR2 (Input to REPLY, SEND)

(Output from REPLY)

This field specifies the address of the area to contain the reply text of the message.

As input to SEND, this field identifies the area into which IUCV places the reply text.

As input to REPLY, this field identifies the area from which IUCV takes the reply text.

The REPLY function updates the contents of this field. REPLY increases the buffer address by the length of the REPLY moved.

IPBFLN1

IPBFLN1F (Input to RECEIVE, SEND)

(Output from DESCRIBE, RECEIVE)

(Reflected in the pending-message IUCV external interrupt)

As an input, this field specifies the length of the input buffer (IPBFADR1 field). The label IPBFLN1 is used by the IUCV macro whenever a halfword value is desired, and IPBFLN1F is used whenever a fullword value is desired. The high-order halfword of the IPBFLN1F field is cleared to zeros by the IUCV macro when halfword values are desired. If the macro is not used, it is the user's responsibility to clear this field when using halfword lengths.

As an output, this field indicates the length of the message as follows:

For DESCRIBE, IUCV stores the actual length of the message in this field.

For RECEIVE, if the buffer was exactly the correct length, IUCV stores a zero in this field. If the buffer was too long, IUCV stores the number of bytes remaining in the buffer (that is, the amount that the buffer exceeded the length of the data) in this field; IUCV sets a return code of 0. If the buffer was too short, IUCV stores the residual count in this field (that is, the number of bytes remaining in the message that would not fit into the buffer) and IUCV sets a nonzero return code.

When an external interrupt occurs, this field contains the actual length of the message.

IPBFLN2

IPBFLN2F (Input to REPLY, SEND)

(Output from DESCRIBE, RECEIVE, REPLY, TESTCMPL)

(Reflected in these IUCV external interrupts: pending message, message complete.)

As an input, this field specifies the length of the input buffer (IPBFADR2 field). The label IPBFLN2 is used by the IUCV macro whenever a halfword value is desired, and IPBFLN2F is used whenever a fullword value is desired. The high-order half word of the IPBFLN2F field is cleared to zeros by the IUCV macro when halfword values are desired. If the macro is not used, it is the user's responsibility to clear this field when using halfword lengths.

As an output, this field indicates the length of the message as follows:

For DESCRIBE, IUCV stores the actual length of the reply area in this field.

For RECEIVE, IUCV stores the actual length of the reply area in this field.

For REPLY, if the buffer was exactly the correct length, IUCV stores a zero in this field. If the buffer was too long, IUCV stores the number of bytes remaining in the buffer (the amount that the buffer exceeded the length of the data) in this field and IUCV sets a return code of 0. If the buffer was too short, the IUCV stores the residual count in this field (that is, the number of bytes remaining in the message that would not fit into the buffer).

For TEST COMPLETION, if the buffer was exactly the correct length, IUCV stores a zero in this field. If the buffer was too long, IUCV stores the number of bytes remaining in the buffer (the amount that the buffer exceeded the length of the data) in this field and IUCV sets a return code of 0. If the buffer was too short, IUCV stores the residual count in this field (the number of bytes remaining in the message that would not fit into the buffer) and sets the IPADRPLE bit in the audit trail.

On an external interrupt, this field contains the actual length of the reply or reply area.

For a complete message external interrupt, if the buffer was exactly the correct length, IUCV stores a zero in this field. If the buffer was too long, IUCV stores the number of bytes remaining in the buffer (the amount that the buffer exceeded the length of the data) in this field and IUCV sets a return code of 0. If the buffer was too short, IUCV stores a residual count in this field (that is, the number of bytes remaining in the message that would not fit into the buffer) and sets the IPADRPLE bit in the audit trail.

IPFCNCD (Input to CONNECT)

(Reflected in the pending-connection IUCV external interrupt.)

As an input, this field indicates which CP system service is invoking the CONNECT function. Each supported CP service is identified by

a one-byte numerical code. VM/SP presently supports IUCV communication for only one CP service, SNA virtual console communication services. SNA has a code of zero.

On an external interrupt, this field is valid only when the IUCV function is invoked from CP system code. (The IPVMID field contains 'SYSTEM'.)

IPFLAGS1 (Input to ACCEPT, CONNECT, PURGE, QUIESCE, RECEIVE, REJECT, REPLY, RESUME, SEND, SEVER, TESTCMPL)

(Output from ACCEPT, CONNECT, DESCRIBE, SEND, RECEIVE, REPLY, PURGE, TESTCMPL)

(Reflected in these IUCV external interrupts: pending connection, connection complete, pending message, message complete.)

As an input, this field specifies options for the function requested. As output or on an external interrupt, this field returns specific information about the message or connection. Each bit is treated separately.

Bits not defined as input for a particular function are reserved for that function and should be set to zero.

IPRMDATA (X'80')

(Input to ACCEPT, CONNECT, SEND, REPLY)

(Output from DESCRIBE, RECEIVE, TESTCMPL)

As an input, or a connection-pending or connection-complete external interrupt, this bit set to 1 indicates the communicator is prepared to handle messages using the DATA=PRMMSG option.

When used with a SEND, a REPLY, a message-pending external interrupt, or message-complete external interrupt, this bit set to 1 indicates that the buffer/parmlist contains the message data in the IPRMMSGx fields.

IPFGMCL (X'01')

(Input to PURGE, RECEIVE, REJECT, TESTCMPL)

(Output from DESCRIBE, RECEIVE)

As an input, this bit indicates that a message class (source message class for PURGE and TESTCMPL, target message class for RECEIVE and REJECT) has been specified in field IPSRCCLS or IPTRGCLS. This bit is set by the IUCV macro when the SRCCLS= or TRGGCLS= parameter is specified on the macro.

IUCV sets this bit to 1 as output from the DESCRIBE function so the resulting parameter list is valid input to the next function (normally RECEIVE or REJECT).

When only part of the message data could be received, IUCV sets this bit to 1 as output from the RECEIVE function. IUCV sets this bit so that the resulting parameter list is valid input to a subsequent RECEIVE.

IPFGPID (X'02')

(Input to PURGE, RECEIVE, REJECT, TESTCMPL)

(Output from DESCRIBE, RECEIVE)

As an input, this bit indicates that a path ID has been specified in field IPPATHID. This bit is set by the IUCV macro when the PATHID= parameter is specified on the macro.

IUCV sets this bit to 1 as output from the DESCRIBE function so the resulting parameter list is valid input to the next function (normally RECEIVE OR REJECT).

IUCV sets this bit to 1 as output from the RECEIVE function when only part of the message data could be received. IUCV sets this bit so that the resulting parameter list is valid input to a subsequent RECEIVE.

IPFGMID (X'04')

(Input to PURGE, RECEIVE, REJECT, TESTCMPL)

(Output from DESCRIBE, RECEIVE)

As an input, this bit indicates that a message id has been specified in field IPMSGID. This bit is set by the IUCV macro when the MSGID= parameter is specified on the macro.

IUCV sets this bit to 1 as output from the DESCRIBE function so the resulting parameter list is valid input to the next function (normally RECEIVE or REJECT).

When only part of the message data could be received, IUCV sets this bit to 1 as output from the RECEIVE function. IUCV sets this bit so that the resulting parameter list is valid input to a subsequent RECEIVE.

IPNORPY (X'10')

(Input to SEND)

(Output from DESCRIBE, PURGE, RECEIVE)

(Reflected in the pending-message IUCV external interrupt.)

As an input, this bit indicates, when set to one, that this is a one-way transaction. When the target invokes the RECEIVE function for this message, IUCV queues the MSGBLOK representing this message on the source communicator's reply queue. No reply by the target is

allowed. If this bit is zero, it indicates a two-way transaction. The message is queued on the source's reply queue only when the target invokes the REPLY function for this message.

As an output or on an external interrupt, this bit indicates, when set to one, that this message does not take a reply.

IPPRTY (X'20')

(Input to ACCEPT, CONNECT, REPLY, SEND)

(Output from DESCRIBE, PURGE, RECEIVE, TESTCMPL)

(Reflected in these IUCV external interrupts: pending connection, pending message, message complete.)

As an input to CONNECT, this bit indicates, when set to one, that the source wishes to establish a path that can handle priority communications. When invoked from a virtual machine, priority must also be authorized in the IUCV directory control statement. When invoked from CP system code, this bit is always set to one.

As an input to SEND and REPLY, this bit indicates, when set to one, that this message or reply, is a priority message or reply. If this path was established to handle priority communications, the message is handled as a priority message. If the path cannot handle priority messages, IUCV generates a nonzero return code.

As an output or on an external interrupt for pending message or message complete, this bit indicates that the message is a priority message.

IPQUSCE (X'40')

(Input to ACCEPT, CONNECT)

(Reflected in these IUCV external interrupts: pending connection, connection complete.)

As an input, this bit indicates, when set to one, that the communicator wishes to establish a quiesced path. The other communicator is not able to send messages on a quiesced path. The path can be restored to full communication capability by invoking the RESUME function.

On an external interrupt, this bit indicates, when set to one, that the path is quiesced. The path must be resumed by the communicating partner before messages can be initiated by this user.

IPALL (X'80')

(Input to QUIESCE, RESUME, SEVER)

When this bit is 1, IUCV performs the specified function on all paths for this virtual machine.

If this bit is specified, IUCV ignores the IPPATHID field.

IPMASK (Input to SETMASK)

This field specifies the mask byte to determine which, if any, of the IUCV interrupts a virtual machine is enabled for.

The SETMASK function cannot be invoked from CP system code.

The bits defined for IUCV are:

| | | |
|---|---|---|
| IPSNDN | X'80' | Enable for nonpriority messages |
| IPSNDP | X'40' | Enable for priority messages |
| IPRPYN | X'20' | Enable for nonpriority replies |
| IPRPYP | X'10' | Enable for priority replies |
| IPCTRL | X'08' | Enable for IUCV control interrupts (CONNECT, SEVER, ACCEPT, QUIESCE, RESUME) |

IPMSGID (Input to PURGE, RECEIVE, REJECT, REPLY, TESTCMPL)

(Output from DESCRIBE, PURGE, RECEIVE, REJECT, SEND, TESTCMPL)

(Reflected in these IUCV external interrupts: pending message, message complete.)

As an input, this field specifies the message identifier of the message to search for. The message identifier uniquely identifies a particular message. It is generated by IUCV and returned by the SEND function when the message is created.

This field is an optional input to the functions listed above. When this field is specified, the path id and message class (IPSRCCLS for PURGE and TESTCMPL, IPTRGCLS for RECEIVE, REJECT and REPLY) must also be supplied.

When this field is used for the above functions, the bit IPFGMID field of IPFLAGS1 must be set to 1.

As an output or on an external interrupt, this field indicates the message id of the message associated with this function or interrupt.

IPMSGLIM (Input to ACCEPT, CONNECT)

(Output from ACCEPT, CONNECT)

(Reflected in the pending-connection IUCV external interrupt.)

As an input, this field specifies the limit of outstanding messages to be allowed on the path established by this CONNECT. A message limit can also be specified on the IUCV directory control statement. If message limit has been specified in the directory for this user, you may not specify a value larger than the directory specification with this parameter of the IUCV macro.

The maximum value that can be specified is 255. For CP system code, there is no overriding directory value. If this field contains a zero, IUCV assumes a default of 10.

As an output or on an external interrupt, this field contains the message limit for this path.

IPMSGTAG (Input to SEND)

(Output from PURGE, TESTCMPL)

(Reflected in the message-complete IUCV external interrupt.)

This field specifies the tag data of the message created by invoking the SEND function. IUCV returns the message tag when the message completes. The source communicator can use this field to tie an incoming message-complete interrupt or output of TESTCMPL to the original SEND request.

As an output or on an external interrupt, this field indicates the message tag of the message associated with this function or interrupt.

IPPATHID (Input to ACCEPT, PURGE, QUIESCE, RECEIVE, REJECT, REPLY, RESUME, SEND, SEVER, TESTCMPL)

(Output from CONNECT, DESCRIBE, PURGE, RECEIVE, REJECT, TESTCMPL)

(Reflected in these IUCV external interrupts: pending connection, connection complete, pending message, message complete, sever, quiesce, resume.)

This field specifies the path identifiers associated with a message. IUCV assigns a path identification and returns the value in the CONNECT parameter list. All further communications on a path must specify the PATHID that was returned from CONNECT. PATHIDs are sequential from X'0000' to the maximum connections allowed for this virtual machine. As paths are severed, IUCV reuses the vacated PATHIDs.

If you specify MSGID on the PURGE, RECEIVE, REJECT, or TESTCMPL functions, IUCV requires that you specify PATHID and message class (IPSRCCLS or IPTRGCLS, as appropriate).

This field is ignored if the IPALL bit in IPFLAGS1 is set to one.

When this field is used on the PURGE, RECEIVE, REJECT, and TESTCMPL functions, the IPFGPID bit must be set to 1 in the IPFLAGS1 field. This bit is set by the IUCV macro when the PATHID= function is specified on the macro.

As an output or on an external interrupt, this field indicates the pathid of the message associated with this function or interrupt.

IPRCODE (Output from ACCEPT, CONNECT, DCLBFR, DESCRIBE, PURGE, QUIESCE RECEIVE, REJECT, REPLY, RESUME, SEND, SEVER, TESTCMPL)

IUCV places a value in this field only when an error is encountered in processing a function. The contents of this field are function dependent. The possible values for this field are listed in Figure 24 on page 171, "IUCV Return Codes and Completion Codes."

Only one error is returned from any function. IUCV terminates the function when the first error is encountered.

IPRMMSGx (Input to SEND, REPLY)

(Output from DESCRIBE, RECEIVE, TESTCMPL)

(Reflected in these IUCV external interrupts: message-pending, message complete.)

For SEND and REPLY, these fields specify the parameter list data. IPRMMSG is two fullwords in length, shown as IPRMMSG1 and IPRMMSG2.

IPSRCCLS (Input to PURGE, SEND, TESTCMPL)

(Output from PURGE, TESTCMPL)

(Reflected in the message-complete IUCV external interrupt.)

This field specifies the source message class associated with a message.

For PURGE and TESTCMPL, this field optionally specifies the source message class of the message to be purged or completed. If omitted, IUCV does not use the source message class in the search for the message.

For SEND, this field specifies the source message class that IUCV stores in the MSGBLOK that represents the message.

As an input to the PURGE and TESTCMPL functions, the IPFGMCL bit must be set in the IPFLAGS1 field. This bit is set by the IUCV macro when the SRCCLS= parameter is specified on the macro.

As an output or on an external interrupt, this field indicates the source message class of the message associated with this function or interrupt.

IPTRGCLS (Input to RECEIVE, REJECT, REPLY, SEND)

(Output from DESCRIBE, RECEIVE, REJECT)

(Reflected in the pending-message IUCV external interrupt.)

This field specifies the target message class associated with this message.

For RECEIVE and REJECT, this field optionally specifies the target message class of the message to be received or rejected. If omitted, IUCV does not use the target message class in the search for the message.

For SEND, this field specifies the target message class that IUCV stores in the MSGBLOK representing the message.

For REPLY, this field specifies the target message class of the message being responded to.

As input to the RECEIVE or REJECT functions, the IPFGMCL bit in the IPFLAGS1 field must be set to 1. This bit is set by the IUCV macro when the TRGCLS= parameter is specified on the macro.

As an output or on an external interrupt, this field indicates the target message class of the message associated with this function or interrupt.

IPTYPE (Reflected in these IUCV external interrupts: pending connection, connection complete, pending message, message complete, sever, quiesce, resume.)

This field indicates the type of external interrupt that is being reflected. The values that are found in this field and their meanings are:

01 - Pending connection

02 - Connection complete

03 - Path has been severed

04 - Path has been quiesced

05 - Path has been resumed

06 - Pending priority message completion

07 - Pending nonpriority message completion

08 - Pending priority message

09 - Pending nonpriority message

IPUSER (Input to ACCEPT, CONNECT, QUIESCE, RESUME, SEVER)

(Reflected in these IUCV external interrupts: pending connection, connection complete, sever, quiesce, resume.)

As an input, this field specifies the 16 byte user data IUCV reflects to the target.

On an external interrupt, this field contains the data specified by the communicating partner.

IPVMID (Input to CONNECT)

(Reflected in the pending connection IUCV external interrupt.)

As an input, this field specifies the eight-character userid of the virtual machine or CP system service to which you want to establish this path.

On an external interrupt, this field contains the ID of the virtual machine that issued the CONNECT. This field contains 'SYSTEM' if the CONNECT was issued by CP system code.

| IUCV FUNCTION | RETURN CODES (Returned in IPRCODE) | CONDITION CODES CC= |
|---|---|---|
| ACCEPT | 00 - Normal return<br>01 - Invalid path id - not a pending connection<br>18 - Value in IPMSGLIM exceeds 255<br>20 - Connection cannot be completed - originator has invoked SEVER | 0 - Normal completion - external interrupt queued to notify originator<br>1 - Nonzero value stored at IPRCODE |
| CONNECT | 00 - Normal return<br>11 - Target communicator is not logged on<br>12 - Target communicator has not invoked the DECLARE BUFFER function<br>13 - Maximum number of connections for this communicator exceeded<br>14 - Maximum number of connections for target exceeded<br>15 - No authorization found<br>16 - Invalid CP system service name<br>17 - Invalid function code in IPFCNCD<br>18 - Value in IPMSGLIM exceeds 255 | 0 - Normal completion - partial connection established. External interrupt queued to notify target of pending connection<br>1 - Nonzero value stored at IPRCODE |
| DECLARE BUFFER | 00 - Normal return<br><br>19 - A buffer has been previously declared | 0 - Normal completion<br>1 - Nonzero value stored at IPRCODE<br>3 - Errors encountered in reading directory |
| DESCRIBE | 00 - Normal return | 0 - Normal completion<br>2 - No message found |

**Figure 24 (Part 1 of 3). IUCV Return Codes and Completion Codes**

| IUCV FUNCTION | RETURN CODES (Returned in IPRCODE) | CONDITION CODES CC= |
|---|---|---|
| PURGE | 00 - Normal return<br>01 - Invalid path id<br>08 - Message found but message class invalid | 0 - Normal completion<br>1 - Nonzero value stored at IPRCODE<br>2 - No message found |
| QUERY | None | 0 - Normal completion<br>3 - Errors were encountered reading directory |
| QUIESCE | 00 - Normal return<br>01 - Invalid path id specified | 0 - Normal completion<br>1 - Nonzero value stored at IPRCODE |
| RECEIVE | 00 - Normal return<br>01 - Invalid path id<br>05 - Receive buffer too short to contain message<br>06 - Fetch protection exception on send buffer<br>07 - Addressing exception on send buffer<br>08 - Message id found but message class or path id invalid<br>09 - Message has been purged<br>10 - Message length is negative | 0 - Normal completion<br>1 - Nonzero value stored at IPRCODE<br>2 - No message found |
| REJECT | 00 - Normal return<br>01 - Invalid path id<br>08 - Message id found but message class or path id invalid | 0 - Normal completion<br>1 - Return code stored<br>2 - No message found |
| REPLY | 00 - Normal return<br>01 - Invalid path id<br>05 - Answer buffer too short to contain message<br>06 - Storage protection exception on answer buffer<br>07 - Addressing exception on answer buffer<br>08 - Message id found but message class or path id invalid<br>09 - Message has been purged<br>10 - Message length is negative<br>21 - Parameter list data not allowed on this path | 0 - Normal completion<br>1 - Nonzero value stored in IPRCODE<br>2 - No message found |

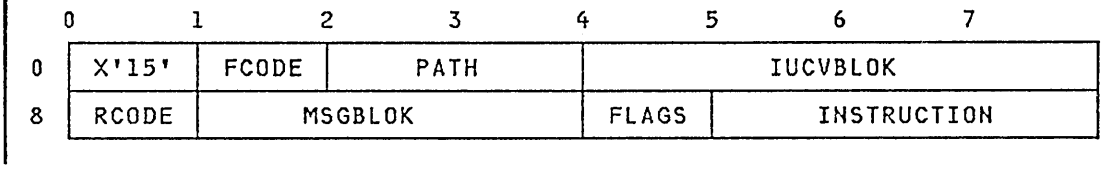
**Figure 24 (Part 2 of 3). IUCV Return Codes and Completion Codes**

| IUCV FUNCTION | RETURN CODES (Returned in IPRCODE) | CONDITION CODES CC= |
|---|---|---|
| RESUME | 00 - Normal return<br>01 - Invalid path id specified | 0 - Normal completion<br>1 - Nonzero value stored at IPRCODE |
| RETRIEVE BUFFER | None | 0 - Normal completion |
| SEND | 00 - Normal return<br>01 - Invalid path id<br>02 - Path quiesced - no sends allowed<br>03 - Message limit exceeded<br>04 - Priority message not allowed on this path<br>10 - Message length is negative<br>21 - Parameter list data not allowed on this path | 0 - Normal completion<br>1 - Nonzero value stored at IPRCODE |
| SET MASK | 00 - Normal return | 0 - Normal completion |
| SET CONTROL MASK | 00 - Normal return | 0 - Normal completion |
| SEVER | 00 - Normal return<br>01 - Invalid path id specified | 0 - Normal completion<br>1 - Nonzero value stored at IPRCODE |
| TEST COMPLETION | 00 - Normal return<br>01 - Invalid path id<br>08 - Message id found but message class or path id invalid | 0 - Normal completion<br>1 - Nonzero value stored at IPRCODE<br>2 - No message found<br>3 - Nonzero audit trail stored |
| TEST MESSAGE | None | 1 - Messages queued on the Send queue<br>2 - Messages queued on the Reply queue<br>3 - Both messages and replies are queued |

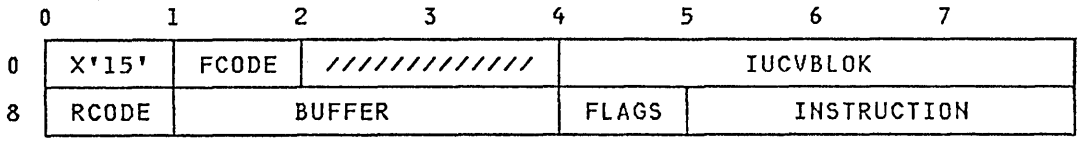
**Figure 24 (Part 3 of 3). IUCV Return Codes and Completion Codes**

## IUCV Trace Table Entry Formats

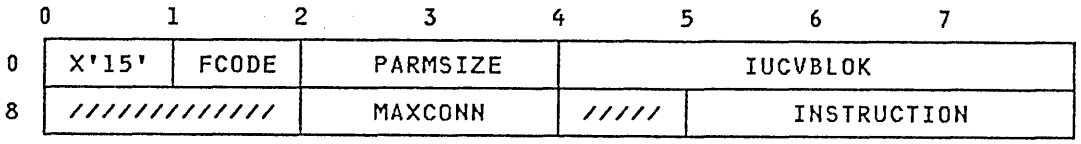
ACCEPT, CONNECT, DESCRIBE, PURGE, QUIESCE, RECEIVE, REJECT, REPLY, RESUME, SEND, SEVER, TESTCMPL



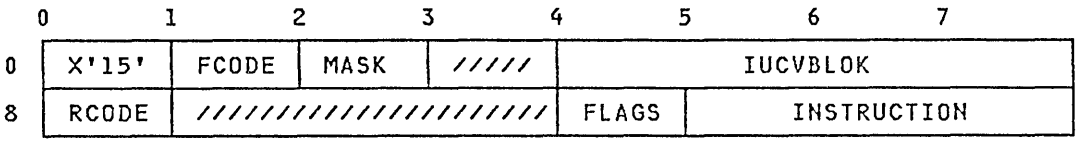
DCLBFR, RTRVBFR



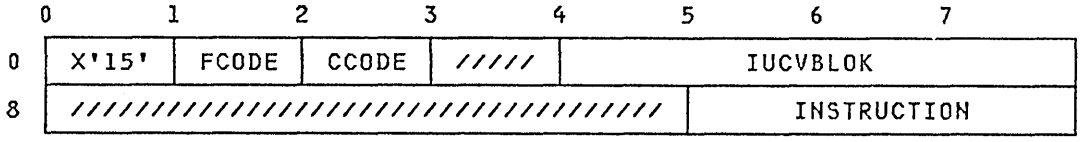

QUERY




SETMASK, SETCMASK




TESTMSG




## *Trace Table Entry Field Definitions*

This section explains, for each IUCV trace table field, the functions for which this field is valid, and the meaning of the field.

BUFFER (Used on DCLBFR, RTRVBFR)

This field contains the virtual buffer address specified by the user for IUCV external interrupt information.

CCODE (Used on TESTMSG)

This field contains the condition code returned to the invoker of the TEST MESSAGE function if a message was pending at the time the TEST MESSAGE function was issued. If no message is pending when the TEST MESSAGE is issued, this field contains zero. Bits 6 and 7 of this CCODE field are used for the condition code.

FCODE (Used on all entries)

This field indicates the exact function executed. One of the following function codes is found in this field.

```
X'00' - QUERY          X'09' - PURGE
X'01' - TESTMSG        X'0A' - ACCEPT
X'02' - RTRVBFR        X'0B' - CONNECT
X'03' - DESCRIBE       X'0C' - DCLBFR
X'04' - SEND           X'0D' - QUIESCE
X'05' - RECEIVE        X'0E' - RESUME
X'06' - REPLY          X'0F' - SEVER
X'07' - TESTCMPL       X'10' - SETMASK
X'08' - REJECT         X'11' - SETCMASK
```

FLAGS

(Used on ACCEPT, CONNECT, DCLBFR, DESCRIBE, PURGE, QUIESCE, RECEIVE, REJECT, REPLY, RESUME, SEND, SETCMASK, SETMASK, SEVER, TESTCMPL)

This field is a copy of the input flags specified by the user in the field IPFLAGS1 of the parameter list. Note that the use of these flags varies by function and that the user may have set flags that are not used by the function.

INSTRUCTION

(Used on all entries)

This field contains the address of the instruction following where the function was invoked. This address is a real address if the IPCPENTY (X'08') bit in the field FLAGS is set to one. Otherwise, the address is an address in a virtual machine.

IUCVBLOK

(Used on all entries)

This field contains the address of the IUCVBLOK associated with the invoker. For the QUERY function, this field may be zero if no IUCVBLOK currently exists for the invoker. For the DECLARE BUFFER function, this field contains the address of the IUCVBLOK created by this function.

MASK

(Used on SETMASK, SETCMASK)

This field contains a copy of the mask field that was specified by the virtual machine.

MAXCONN

(Used on QUERY)

This field contains the maximum number of connections allowed by the virtual machine issuing this request.

MSGBLOK

(Used on DESCRIBE, PURGE, RECEIVE, REJECT, REPLY, SEND, TESTCMPL)

This field contains the real address of the MSGBLOK processed by this request.

PARMSIZE

(Used on QUERY)

This field contains the size of IUCV parameter list returned to the invoker of the QUERY function.

PATH

(Used on ACCEPT, CONNECT, DESCRIBE, PURGE, QUIESCE, RECEIVE, REJECT, REPLY, RESUME, SEND, SEVER, TESTCMPL)

This field contains the path id of the path associated with this request. For the CONNECT function, this is the path id associated with the path being created. For the other functions, this is the path id used to process the request.

RCODE

(Used on ACCEPT, CONNECT, DCLBFR, DESCRIBE, PURGE, QUIESCE, RECEIVE, REJECT, REPLY, RESUME, SEND, SETCMASK, SETMASK, SEVER, TESTCMPL)

This field contains the code returned in the field IPRCODE of the parameter list. If this return code field is non-zero, only the TYPE, FCODE, INSTRUCTION, FLAGS, and IUCVBLOK fields are valid. The other fields may be invalid due to the nature of the return code. Invalid fields always contain zeroes.

# SNA Virtual Console Communication Services

SNA Virtual Console Support provides full VM/SP console capabilities to terminal operators on SNA terminal devices and allows the VM/SP user to use SNA terminals as virtual operator consoles.

SNA Virtual Console Communications Services support the following console functions:

- CP/CMS command processing capabilities
- System product or CMS editor processing mode
- Full screen support for 3270 type terminal devices

This support is provided through a virtual service machine (VSM) and the SNA Console Communications Services (SNA CCS) in CP. The VM/VTAM Communications Network Application (VM/VCNA) program product (program number 5735-RC5) running in the virtual service machine acts as the interface between CP and the SNA network. Similarly, the SNA Console Communications Services provides the necessary interface between the existing CP system's console services and the VM/VTAM Communications Network Application (VM/VCNA).

The screen management services are divided between the VM/VTAM Communications Network Application (VM/VCNA) and the SNA Console Communications Services (SNA CCS). VM/VCNA is responsible for the physical screen management and therefore, the device dependent characteristics. Thus, VM/VCNA handles such things as screen size and redisplay of the input line at the terminal. SNA CCS is responsible for logical screen management and thus remains relatively device independent. SNA CCS also passes the terminal input to CP and reflects status and actions to and from the rest of the CP system.

## System Structure

Figure 25 on page 178 illustrates a VM/SP system with the SNA virtual console support. The VTAM service machine (VSM) consists of an SCP, either VS1 or DOS with External Interrupt Services (EIS), VTAM, and VM/VCNA.

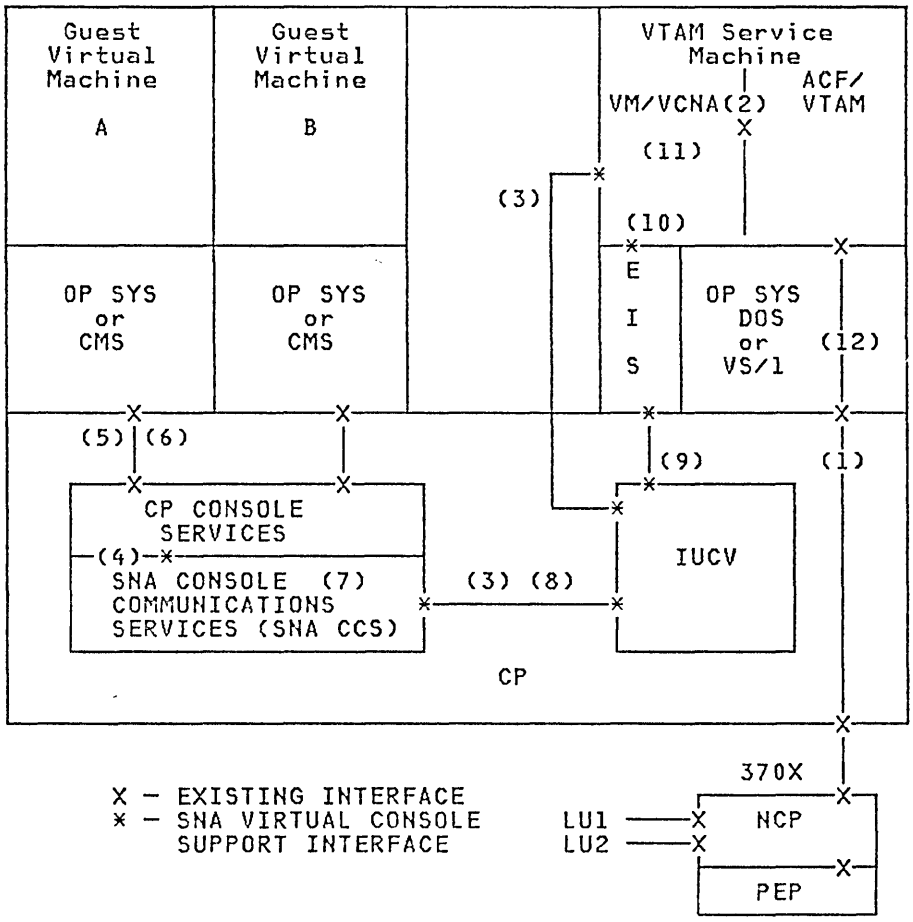



**Figure 25. Virtual Console Support in CP**

1. SNA CCS supports the SNA terminals (LU1, LU2) as virtual consoles. These SNA terminals are attached to a 370X dedicated to the VTAM service machine. Data entered at the terminal goes through its normal path of the NCP, CP, the SCP and VTAM. The guest virtual machine interface to CP is the S/370 architecture provided by virtual machine simulation.

2. VM/VCNA interfaces to VTAM via the standard Application Program Interface (API) in order to perform physical I/O to/from the SNA devices.

3. The terminal input is communicated to the SNA CCS via the Inter-User Communication Vehicle (IUCV) SEND, RECEIVE, and REPLY protocols.

4. SNA CCS receives the interface control block (Work Element Block) with the terminal input data. It interprets the control information that describes the screen environment and the user's actions, and determines the action to be reflected to CP. SNA CCS edits the input line, and passes it to CP along with the action required.

5. SNA CCS either processes the line or sends it to the guest virtual machine for processing.

6. Guest virtual machines request console I/O via the Start I/O interface (SIO) or via the VM/SP Diagnose X'58' facility.

7. SNA CCS intercepts the I/O request and performs logical screen management. A Work Element Block is built to inform VM/VCNA of the action to be initiated on the screen and to hold the output line.

8. SNA CCS uses the IUCV SEND, RECEIVE, and REPLY protocols to communicate the work transaction to VM/VCNA.

9. The IUCV request from SNA CCS to VM/VCNA in the VTAM Service Machine is intercepted by the External Interrupt Services (EIS) in the guest SCP (OS/VS1 with Basic Programming Extensions, or DOS/VSE with the latest release of VSE/AF).

10. EIS notifies VM/VCNA of the incoming message.

11. VM/VCNA receives the Work Element Block, interprets the orders, and performs the physical screen management for the SNA terminal.

12. VM/VCNA causes VTAM to perform the I/O and the output once again goes through its normal path of VTAM, the SCP, CP, and the NCP to get to the SNA user terminal.

## Environments Supported

SNA CCS and the VM/VTAM Communications Network Application (VM/VCNA) handle three 'environments' for the purposes of screen management: console mode, CMS mode, and full screen support mode. These environments represent the interfaces that CP supports for console services to a virtual user terminal and a guest virtual machine (GVM).

1. Console mode is communications between a display operator and either CP or an operating system in a virtual machine (CMS or another operating system). In this mode, the screen is divided into three areas, (input, output, and status), and data to the output area is always directed to the next available line. Console mode I/O is generated when a guest virtual machine issues an SIO to the 3215 user console or CP generates console I/O requests internally in response to CP commands.

2. CMS mode is DIAGNOSE X'58', CCW op code X'19' transactions. In this mode, the CMS editor or an application program directs output to specific lines on the screen. As with console mode, the screen is divided into three areas (input, output, and status).

3. Full screen support mode (FSSM) is the environment where the display screen is under control of a full screen application program. In this mode, the format of the screen is under application program control and the application program provides all 3270 orders. The interface to CP from a guest virtual machine is DIAGNOSE X'58', CCW op code X'29' or X'2A', for a full screen write or read.

## Processing Descriptions

The following sections describe SNA CCS processing.

### Screen Management

In non-SNA processing, DMKGRF handles the console support for local 327X and 3066 devices. DMKRGA and DMKRGB contain the support for remote devices. These modules perform both the logical and physical screen management needed for the graphic display and printer keyboard terminals.

In SNA processing, in order to support a virtual console for a VTAM service machine terminal user, virtual console support has been divided between the VM/VTAM Communications Network Application (VM/VCNA) and SNA Console Communications Services (SNA CCS)

SNA CCS handles this SNA environment for CP via modules DMKVCP, DMKVCR, DMKVCT, DMKVCV, and DMKVCX. VM/VCNA handles the physical, device-dependent characteristics of the screen, setting up the I/O, and maintaining the current state of the screen. VM/VCNA uses VTAM to perform the I/O. SNA CCS handles the logical control of the screen, directs the VM/VCNA actions, and serves as the interface between the VTAM machine and the existing CP console function support. SNA CCS and VM/VCNA communicate via IUCV.

Modules DMKVCP, DMKVCR, DMKVCT, DMKVCV, and DMKVCX perform the logical functions for CP that are described above. As with non-SNA processing, DMKGRF processes the local 327X/3066 devices, and DMKRGA and DMKRGB support the remote devices.

Note that the logical units supported by VM/VCNA are independent of CP; they cannot be mapped to any real device defined to CP (that is, they are not defined in the RDEVICE macro). SNA CCS provides the necessary interface to make the SNA terminal appear to be a real CP device.

### Communication Interfaces

To communicate, VM/VCNA and SNA CCS pass a work element block (WEBLOK) between themselves. The WEBLOK contains the transaction orders for the other component (SNA CCS or VM/VCNA), the environment, and the data for the CP system or the user's terminal. See the section "Work Element Block" that follows or see *VM/SP Data Areas and Control Block Logic, Volume 1* for a detailed description of the WEBLOK.

SNA CCS and VM/VCNA communicate via the Inter-User Communication Vehicle (IUCV). Figure 26 on page 181 illustrates the interfaces used in SNA processing. DMKQCN presents requests from CP, CMS or, a guest virtual machine for terminal writes to SNA CCS via CONTASKS. DMKQCO presents requests from CP, CMS or, a guest virtual machine for terminal reads to SNA CCS via CONTASKS. SNA CCS passes input from the SNA terminal to CP and the virtual machines via DMKCFM and DMKVCN. This is the same way DMKGRF handles local terminal support.

In SNA processing, CP handles terminal input and interfaces normally with one exception: CP must use logical unit names, instead of real addresses, to reference SNA terminals.

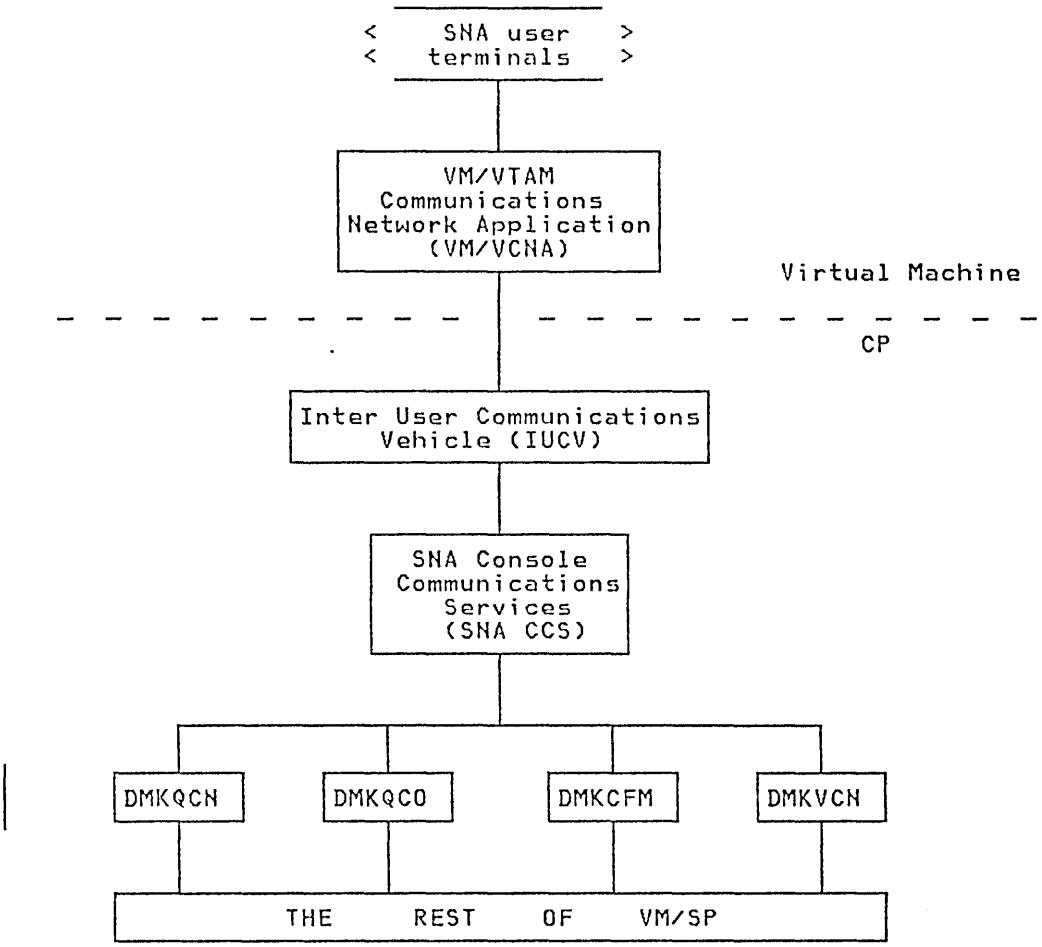



**Figure 26. SNA Virtual Console Support Interfaces**

## Functions

SNA CCS handles the following functions in support of the console, CMS, and full screen mode environments for SNA terminals:

- Connect VTAM service machine and Logical Units
- Logon a Logical Unit
- Request a read
- Request a write
- Process an enter key
- Process a PA1 key
- Process a PA2 key
- Process a PA3 key
- Process a PF key
- Process an Attention Interrupt
- Process a Cursor Back One
- Logoff a Logical Unit
- Process error conditions
- Quiesce and Resume processing for a given logical unit
- Sever a communications path

### Enabling SNA Terminals

CP operator must issue the ENABLE SNA command. The SNA parameter on the ENABLE command enables all SNA devices and has no effect on non-SNA devices. The ENABLE ALL command enables both non-SNA and SNA devices.

In the multiple VTAM service machine environment, the operator may selectively enable or disable any given VTAM service machine by using the userid option on the ENABLE/DISABLE SNA command. The operator cannot enable and disable individual logical units.

### Establishing Communications Links

The VTAM service machine issues an IUCV CONNECT under two separate conditions:

- VTAM Service Machine CONNECT
- VM/VCNA issues a CONNECT via IUCV to establish an initial global connection between VM/VCNA and SNA CCS. This CONNECT notifies SNA CCS that a new VTAM service machine has logged on and is ready to service logical units. If the VM/SP operator has issued the ENABLE SNA or ENABLE ALL command, SNA CCS accepts the CONNECT, and authorizes VM/VCNA to allow users to logon to SNA terminals. SNA CCS creates a VTAM Service Machine Block (VSMBLOK) for that VTAM service machine. In a multiple VTAM service machine environment, the VSMBLOK allows SNA CCS to associate the logged on SNA user with the correct VM/VTAM Communications Network Application (VM/VCNA). See *VM/SP Data Areas and Control Block Logic, Volume 1* for a detailed description of the VSMBLOK.
- Logical Unit CONNECT

  To logon to VM/SP, the SNA terminal user must first logon to VM/VCNA running in the VTAM service machine. To logon to VM/VCNA, the user issues the ACV/VTAM LOGON command. When logging on to VM/VCNA, the terminal user may optionally specify the userid (or the userid and password) of his virtual machine in the DATA portion of the ACF/VTAM LOGON command. The following forms of the LOGON command are valid:

  - LOGON APPLID (VCNA)
  - LOGON APPLID (VCNA) DATA (userid [password])

    If you specify VCNA in the APPLID field, ACF/VTAM queues a logon request to VM/VCNA.

    If no logon data (VM/SP userid and password) is specified, the system writes a VM/370 logo to the terminal under the control of VM/VCNA. From this point on, the user logs on to VM much the same as he does with a local terminal. The attention interrupt generated when the user clears the logo from the screen causes VM/VCNA to issue an IUCV CONNECT SVC on behalf of the terminal. If 'SNA' is still enabled, CCS builds an RDEVBLOK and a SNA Resource Block (SNARBLOK) and chains them to the VSMBLOK built during the VM/VCNA connect described above. This ties the user's control block (SNARBLOK) to the VTAM service

machine the user is logged onto. The system must tie these blocks together since the logical unit's 'LUNAME', which is represented by the SNARBLOK, is unique only to its own VTAM service machine. That is, it is possible to have duplicate lunames among two or more VTAM service machines.

After the connection is established, VM/VCNA and SNA CCS exchange initialization information. VM/VCNA sends luname, device class, device type, line length, pace value (for controlling the number of writes to the screen), model number, and its IUCV path ID for this logical unit and then waits for LOGON processing to complete. SNA CCS initializes the SNARBLOK and RDEVBLOK with the data supplied by VM/VCNA.

If the user specified a userid and password on the ACF/VTAM LOGON, the VM/370 logo is not displayed. VM/VCNA sends the logon data to SNA CCS in response to the first CP read request to enter userid. If the user specified only the userid, CP prompts the terminal user for the password.

- AUTOMATIC LOGON

- The installation may specify "automatic" logon to VM/VCNA for SNA terminals. This can be accomplished in two ways:

  1. The installation can specify LOGAPPL=(VCNA) in the logical unit definition. This causes ACF/VTAM to queue a logon request to VM/VCNA when the logical unit is activated.

  2. The ACF/VTAM operator may issue a VARY ACTIVATE command for the logical unit, specifying VCNA on the LOGON= parameter.

For further information concerning ACF/VTAM LOGON refer to the *IBM ACF/VTAM System Programmer's Guide.*

**Real Device Simulation**

When VM/VCNA connects to SNA CCS for a logical unit, VM/VCNA identifies the SNA logical unit to SNA CCS. In addition, VM/VCNA identifies any device characteristics that CP or CMS need to perform their functions. SNA CCS simulates a real device by dynamically building a Real Device Block (RDEVBLOK) and assigning this RDEVBLOK to the SNA user's virtual machine.

SNA CCS initializes the fields for the RDEVBLOK instead of DMKRIO. In addition, SNA CCS builds a control block for SNA, a SNARBLOK. The SNARBLOK contains the status and control fields for SNA CCS. See *VM/SP Data Areas and Control Block Logic, Volume 1* for a detailed description of the SNARBLOK.

The RDEVBLOK is chained to the VSMBLOK belonging to the VSM that issued the IUCV connect for it, and the RDEVBLOK points to the SNARBLOK for that LU. The RDEVBLOK and SNARBLOK are, however, contiguous in storage. CP references to the RDEVBLOK are still valid in the SNA environment.

As in non-SNA processing, the VMTERM field of the VMBLOK and the VDEVREAL field of the VDEVBLOK point to the RDEVBLOK.

When special SNA processing is necessary, an indicator in the RDEVBLOK (RDEVSNA) denotes that this is a SNA type RDEVBLOK. The RDEVSNA field is an alternate definition for the current RDEVADD field. The real device address has no meaning for SNA logical units.

### Command Handling

After VM/VCNA completes the initial processing for the SNA logical unit, it passes the user's LOGON request to SNA CCS. SNA CCS edits the LOGON command and all subsequent commands and passes them to CP console services using CP interfaces. CP processes the commands the same way it processes non-SNA commands. However, VM/VCNA, rather than CP, manages redisplay of the input line.

### Work Element Block

The work element block serves as the interface between SNA CCS and VM/VCNA. Both SNA CCS in CP and VM/VCNA in the VTAM service machine create work element blocks. In SNA CCS, the work element block is known as the WEBLOK. In VM/VCNA, the work element block is known as the DTIWEB. SNA CCS and VM/VCNA pass the WEBLOK between them and use it as the interface for all requests for work from the other component. The data portion of the work element block contains input or output lines to be passed and the control portion contains transaction orders and environment data. See *VM/SP Data Areas and Control Block Logic, Volume 1* for a detailed description of the WEBLOK.

### Work Element Indicator Block

SNA CCS creates the work element indicator block (WEIBLOK) as a header for the WEBLOK. Its function is to identify a unit of work that is in progress or that has not yet been processed. The WEIBLOK points to the WEBLOK and CONTASK associated with a given user. See *VM/SP Data Areas and Control Block Logic, Volume 1* for a detailed description of the WEIBLOK.

### SNA I/O Processing

For non-SNA processing, CP console services build channel programs, IOBs, and use DMKIOS to perform their I/O. SNA processing moves the physical device management to VM/VCNA. Instead of calling DMKIOS, CP then passes control to SNA CCS. SNA CCS does not build any channel programs or IOBs. It determines what action must be taken for the console and sends the transaction to VM/VCNA instead of to DMKIOS. VM/VCNA and VTAM set up the I/O operations to the terminal and issue an SIO. CP intercepts this SIO and performs the I/O the same way it does for non-SNA processing.

### Redisplay of Input Line

Input line redisplay for SNA terminals is handled by VM/VCNA.

- VM/VCNA Redisplay of input line

  To reduce the number of VTAM SENDs to the terminal, VM/VCNA does not immediately redisplay the input line. Instead, it holds the I/O operation until SNA CCS requests more I/O to that terminal; for example, the response to the input or a message. When VM/VCNA receives a write to the device, it sends the input line to be redisplayed and the information from SNA CCS to be written to the device in one VTAM send.

VM/VCNA clears the input area, updates the status field, and redisplays the line using the same VTAM SEND.

- CCS Redisplay Timer

  VM/VCNA passes a timer variable to SNA CCS when it invokes the IUCV CONNECT function. This value indicates to SNA CCS how long it should wait for a command to complete before SNA CCS issues an IUCV SEND to the VTAM service machine to request input line redisplay.

  SNA CCS sets a timer to tell it how long to wait before requesting redisplay of the line. This is necessary since some commands do not produce any output, and CP or CMS might require a significant amount of time to finish the command processing. If the timer expires before CP has output to write to that terminal, SNA CCS issues an IUCV SEND to VM/VCNA requesting a write for the redisplay.

## TRQBLOK

In non-SNA processing, DMKGRF builds a Timer Request Block (TRQBLOK) that it uses (1) for its status flags (2) for an interrupt return address after an I/O operation (3) after a timer expires and (4) as a header to chain CONTASKS when in FSS mode.

In SNA processing, SNA CCS does not use TRQBLOK for the above functions because: (1) status fields are kept by the VCNA for each SNA terminal user, (2) IUCV mechanisms are used to return control after SEND requests to the VM/VCNA, (3) the timer support for the alarm, MORE/HOLDING state, and NOT ACCEPTED has been moved to VM/VCNA, and (4) SNA CCS has its own control block structure to associate a user with its related CONTASKS, IUCV control blocks, and the work element block. Since the processing of CONTASKs has been streamlined to help performance, the TRQBLOK is no longer needed for queueing CONTASKs.

However, in SNA processing, a TRQBLOK is still created, since a timer is required for the input line redisplay processing described above.

## I/O REQUESTS

DMKGRF, the module that manages I/O to a real 3270, performs requests for I/O from DMKQCN synchronously. When DMKQCN requests a response, DMKGRF schedules an IOB for the I/O operation and waits for the I/O to complete before sending the response. SNA CCS takes the virtual machine out of SIO wait state as soon as the I/O to the real device is started. In the case of a write, SNA CCS sends the write request to VM/VCNA, takes the virtual machine out of the SIO wait state, and returns immediately to the caller with a successful completion response as if the I/O had completed successfully.

In some situations, SNA CCS waits for a response from VM/VCNA before responding with a return code to the CP system. This is governed by a 'pacing value' equivalent to the number of lines for a full screen. In this way, VM/VCNA can reach SNA CCS with a PA1 key indicator to stop processing; SNA CCS does not flood IUCV and VM/VCNA with output from some commands (for example, DISPLAY). SNA CCS always waits for a response from VM/VCNA for DIAGNOSE X'58' writes for CMS and full screen support modes before returning to the caller with a response.

SNA CCS queues a CONTASK if it is waiting for a response on either a write or a read request. It does not split CONTASKs for multiple line writes but passes the entire write buffer to VM/VCNA, thus reducing IUCV SENDs and RECEIVEs.

### VTAM I/O Reduction

SNA CCS batches console function and virtual machine SIO output lines in a 1K byte buffer. The batch lines are sent to VCNA when the buffer is full, a read is initiated by a virtual machine or CP to a SNA terminal, the pace value reaches zero, the redisplay timer expires, a Diagnose X'58' operation takes place, or the virtual machine is dropped from the dispatch queue.

The batching technique and priority structure ensures that either a full screen of information is presented to VM/VCNA for each CP or CMS console transaction or the transaction is complete (for those transactions with less than a full screen of data) prior to control being given to the VSM.

### MORE/HOLDING Condition

To reduce the number of IUCV transactions, VM/VCNA resolves the MORE/HOLDING condition when it occurs on the screen. VM/VCNA takes whatever action is appropriate and avoids notifying SNA CCS of the screen status in most cases. In cases when a mode change may take place (PA1 key) or an interrupt must be reflected to a user's virtual machine (PA1 or PA2 key), VM/VCNA resolves the MORE/HOLDING condition, then notifies SNA CCS of which key was pressed and the screen status at the time. VM/VCNA resolves pressing of the clear key or enter key in MORE/HOLDING status without notifying SNA CCS.

### SNA Accounting

VM/VCNA records accounting data on a terminal user basis. When the SNA user logs off, VM/VCNA passes a maximum of 62 bytes of accounting data, in the WEBLOK, to SNA CCS. SNA CCS uses the CP accounting module, DMKACO to write a VTAM accounting record (type X'07') in the CP accounting file. Neither SNA CCS nor DMKACO are aware of the contents of the VTAM accounting record.

SNA CCS accrues processor time for a terminal user while it is processing for that user. This time is added to the time CP already accumulated for the user. The time appears in the accounting record produced when the user logs off.

**Note:** Refer to *VM/VCNA Installation, Operation, and Terminal Use,* SC27-0502 for details of the accounting records.

### NCP and PEP Sharing

Since CP supports only a back level of NCP that does not support SNA and VTAM loads ACF/NCP, you must prevent CP from loading/reloading the back-level NCP at initialization and at restart. Refer to *VM/SP Planning Guide and Reference* for information on how to accomplish this.

### User Termination

When a user issues the VM/SP LOGOFF or a ACF/VTAM LOGOFF, the control blocks related to the user's virtual machine and SNA terminal are released to free storage. When VM/VCNA issues the SEVER indicating that a user has logged off, SNA CCS need only issue a SEVER for its path. If a SEVER reaches SNA CCS and there is a SNARBLOK that indicates the user is disconnected, the

path is severed. The control blocks are released when the user is eventually logged off. If SNA CCS gets the SEVER and there is a SNARBLOK but the user is not disconnected, then SNA CCS disconnects the virtual machine associated with the SNARBLOK.

#### Shut Down

To shut down the system, the VM/SP system operator should notify users that the system is shutting down. If the SNA operator has the proper class, he can force off any SNA user that did not log off. In this way, VM/VCNA can collect accounting data for its users and record it in CP. The DISABLE SNA (userid) command can be used to prevent additional users from logging on. In this way, VM/VCNA can be stopped without bringing down the VTAM service machine that it is running in. Any other application in the VTAM service machine may continue to run unaffected.

#### VM/VCNA Operator Considerations

While it is possible for the operator of the VTAM service machine to disconnect from a 'local' terminal, extreme care must be exercised. The VM/VCNA operator must 'SET RUN ON' prior to disconnecting from the 'local' terminal. If the operator does not do this, unpredictable results occur and a deadlock of the VM/VCNA is likely.

The operator of the VTAM service machine should never disconnect from the service machine and then reconnect from a SNA logical unit, using the same operator userid that was used for the service machine. That is, the operator must not logon at a terminal that is managed by the VCNA as the VCNA Operator.

## SNA CCS Entries in CP Internal Trace Table

SNA Console Communications Services (SNA CCS) creates trace table entries in the CP Internal Trace Table to leave an audit trail of its activities.

SNA CCS places an entry in the CP trace table for each inbound transaction; SNA CCS creates a trace table entry for each outbound transaction after going to the Inter-User Communication Vehicle (IUCV) to communicate the entry to the VM/VTAM Communications Network Application (VM/VCNA).

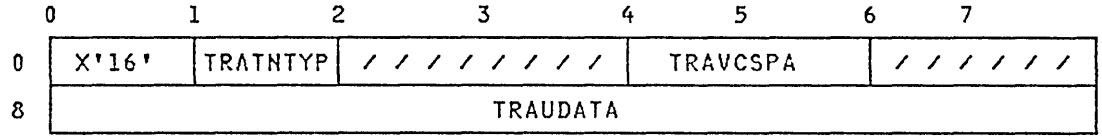
The entry identifies the type of IUCV transaction, the SNA user that initiated the transaction, and the pertinent characteristics of the transaction environment itself. The transaction can be correlated throughout the system by the use of the CCS and VCNA path id's and the IUCV message id. These fields can be matched with corresponding or similar fields in the IUCV trace elements in CP and VCNA trace elements in VTAM.

For an error trace, SNA CCS places an entry in the CP trace table for logical errors and errors on IUCV transmissions. If the WEBLOK that is passed between SNA CCS and VM/VCNA is invalid, the data in the trace element pertains to the invalid WEBLOK.

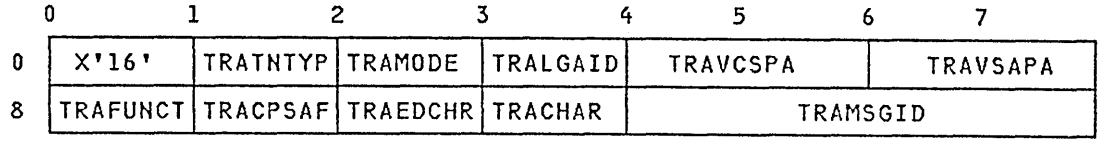
### *Trace Table Entry Formats*

The following tables show the formats of trace table entries created by SNA CCS.

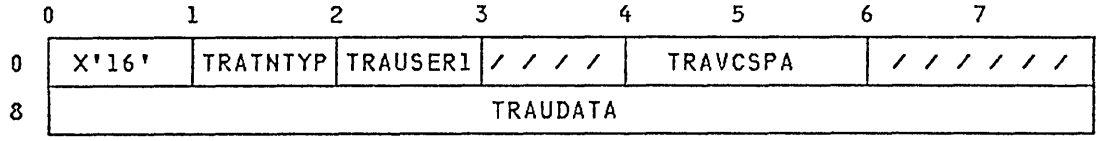
ACCEPT (00) (VTAM service machine and Logical Unit) CONNECT for Logical Unit (12)



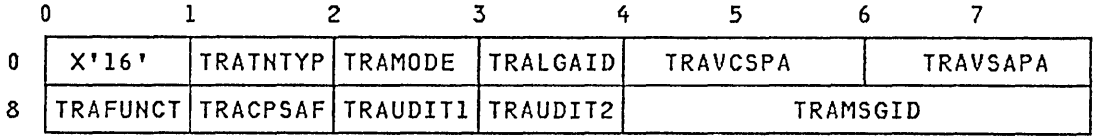

RECEIVE (04), REPLY (06), SEND 1WAY (08), SEND 2WAY (09), LOGIC ERROR in CCS WEBLOK (0B), LOGIC ERROR in VCNA WEBLOK (13)



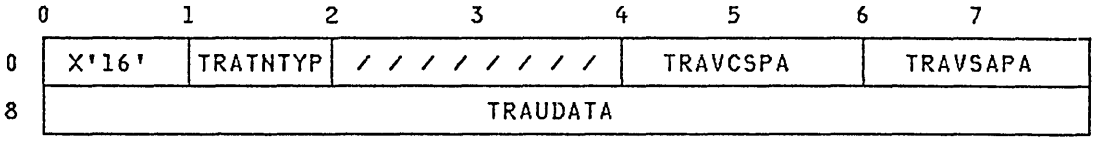

SEVER (0A)



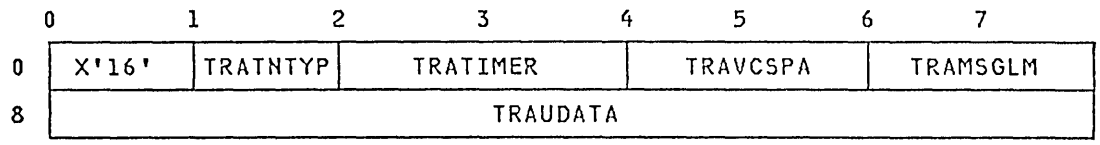

REPLY from VCNA (0C)



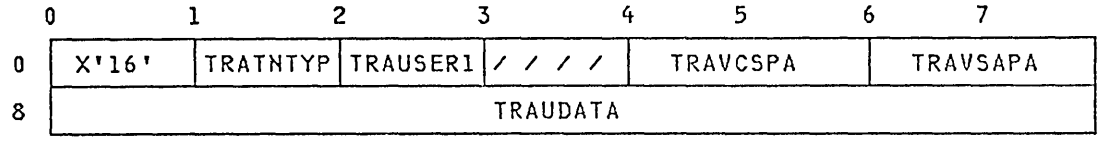

QUIESCE from VCNA (0D), RESUME from VCNA (0F)



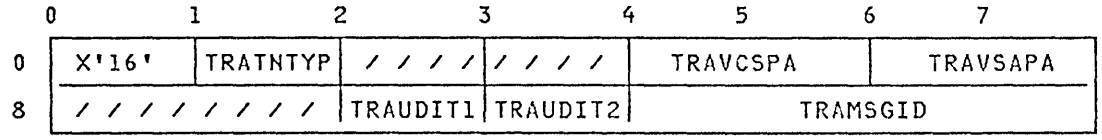
CONNECT for VTAM service machine (0E)



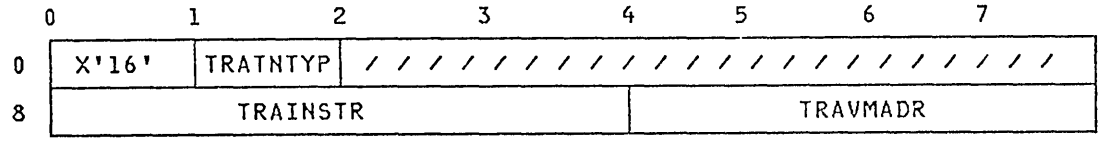

SEVER from VCNA (10)




MESSAGE COMPLETE (11)




ABEND 02 (15)




## *Trace Table Entry Field Definitions*

TRATNTYP Indicates the type of transaction that the trace table entry is for.

**Note:** If the high-order bit is on, this indicates that there was a nonzero return code from IUCV on this transaction. DMKVCXFU writes the trace table entry. The IUCV return code (IPRCODE) is in TRAIPRCD, a one-byte field in the fourth byte of the trace entry.

**Values Defined for TRATNTYP**

| | | |
|---|---|---|
| TRACCEPT | X'00' | ACCEPT |
| TRACNECT | X'01' | CONNECT (not used) |
| TRAQUISC | X'02' | QUIESCE (not used) |
| TRAPURGE | X'03' | PURGE (not used) |
| TRARCEIV | X'04' | RECEIVE |
| TRAREJCT | X'05' | REJECT (not used) |
| TRAREPLY | X'06' | REPLY |
| TRARESUM | X'07' | RESUME |
| TRASEND1 | X'08' | SEND 1 WAY |
| TRASEND2 | X'09' | SEND 2 WAY |
| TRASEVER | X'0A' | SEVER |
| TRAVCSLE | X'0B' | LOGIC ERROR IN CCS WEBLOK |
| TRAVSARP | X'0C' | REPLY FROM VCNA |
| TRAVSAQS | X'0D' | QUIESCE FROM VCNA |
| TRAVSMCN | X'0E' | CONNECT FOR VTAM service machine |
| TRAVSARM | X'0F' | RESUME FROM VCNA |
| TRAVSASV | X'10' | SEVER FROM VCNA |
| TRAVSAMC | X'11' | MESSAGE COMPLETE FROM VCNA - 1 WAY SEND |
| TRALUCON | X'12' | CONNECT FROM VCNA FOR LU |
| TRAVSALE | X'13' | LOGIC ERROR IN VCNA WEBLOK |
| TRAERRSV | X'14' | ERROR IN USER ENVIRONMENT-SEVER USER (not used) |
| TRACTLBK | X'15' | SNA CONTROL BLOCK CHAIN INVALID |

TRAMODE Mode for the transaction (see WEBLOK (WEBMODE)).

TRALGAID Logical mapping of the Attention Identifier (AID) for inbound transactions to CCS (see WEBLOK (WEBLAID)). The field does not have meaning for outbound transactions to VCNA.

TRAUSER1 First byte from the IUCV user data field

TRATIMER Two bytes of timer value from the VSM CONNECT

TRAVCSPA The IUCV path id that identifies the CCS side of the IUCV path for this transaction.

TRAVSAPA The IUCV path id that identifies the VCNA side of the IUCV path for this transaction.

TRAFUNCT Transaction to be performed (see WEBLOK (WEBFUN))

TRACPSAF This field is WEBSAFLG on inbound transactions to CCS and WEBCPFLG on outbound transactions to VCNA. (see WEBLOK (WEBFUN)).

TRAEDCHR Editing characteristics (see WEBLOK (WEBEDIT))

TRACHAR Character set (see WEBLOK (WEBCHAR))

TRAMSGID Message identifier from IUCV IXBLOK

TRAIPRCD IPRCODE from IUCV IPARML for IUCV return code processing

TRAMSGLM IUCV message limit to be specified for CONNECT

TRAUDATA IPUSER from IUCV external interrupt buffer
For inbound: QUIESCE, RESUME, CONNECT for LU
For outbound: ACCEPT

VM userid
For inbound: CONNECT for VTAM service machine
For outbound: SEVER

LUNAME
For inbound: SEVER

TRAUDIT1 IUCV IPAUDIT1 flags from IXBLOK (used for TRAVSARP,TRAVSAMC)

TRAUDIT2 IUCV IPAUDIT2 flags from IXBLOK (used for TRAVSARP,TRAVSAMC)

TRAINSTR Addr of last instruction issued before invoking abend routine (TRACTLBK)

TRAVMADR Current VMBLOK address-used for abend situations (TRACTLBK)

# The Message System Service

The Message System Service is a CP system service. It allows a virtual machine to read incoming messages and responses from CP, as opposed to displaying them on the terminal.

## Establishing Communications

"*MSG" is the assigned Message System Service userid. Communications are established with the Message System Service (*MSG) via IUCV. The IUCV DECLARE BUFFER function is invoked by the virtual machine to allow communications with IUCV, and the IUCV CONNECT function is invoked to establish the communications path to the *MSG system service.

The types of messages that the virtual machine can receive are controlled by specifying the IUCV parameter on the CP SET command. For example, if a user has specified "CP SET MSG IUCV", all messages received via the CP MESSAGE command are sent to the virtual machine via IUCV. IUCV signals the receiving virtual machine with an external interrupt. The message may be retrieved by using the IUCV RECEIVE function and may be used by a program running in the virtual machine.

**Note:** For a complete list of the CP SET commands that can use the IUCV parameter, see *VM/SP CP Command Reference for General Users.*)

The Message System Service identifies the source of the message it intercepts by a code in the IUCV message class field. The message source is interpreted as follows:

| *Class* | *Message Source* |
|---|---|
| 1 | Message sent using CP MESSAGE (MSG) or CP MSGNOH |
| 2 | Message sent using CP WARNING (WNG) |
| 3 | Asynchronous CP messages and CP responses to a CP command executed by a virtual machine using *MSG |
| 4 | Message sent using CP SMSG command |
| 5 | Any data directed to the virtual console by the virtual machine (WRTERM, LINEDIT, etc.) |
| 6 | Error message from CP (EMSG) |
| 7 | Information messages for CP (IMSG) |
| 8 | Single Console Image Facility (SCIF) message from CP. |

Error and information messages (classes 6 and 7) are types of CP messages and are included in class 3 when EMSG and IMSG are not specifically set to IUCV via the CP SET commands.

The format of the data received from IUCV is as follows:

```
col 1     col 9
|         |
V         V
userid    text
```

The userid portion of the data identifies the sender. If the data is not received by means of a MSG, WNG, SMSG, or using SCIF, then the userid is that of the recipient.

If a virtual machine has both a valid path to the *MSG system service and a secondary user specified in the CONSOLE directory control statement (enabling that virtual machine to use SCIF), then incoming messages (except for SMSGs, which are not console messages) are directed to the secondary user instead of the IUCV *MSG system service. If the secondary user is not available, the message is queued on the *MSG system service path.

**Note:** The following types of data are not placed in the console spool file for the indicated conditions:

- CP command output -- if this is being received in a buffer via DIAGNOSE X'08'.
- Messages and Warnings -- if they are being trapped via the IUCV and *MSG System Service.

## DASD Block I/O System Service

The DASD Block I/O System Service is a CP system service. It provides a virtual machine with device-independent access to its virtual DASD devices. Device types supported are the Count Key Data (CKD) devices: 2314, 2319, 3330, 3333, 3340, 3344, 3350, 3375, and 3380, and the Fixed Block Architecture (FBA) devices: 3310 and 3370. (Device 2319 is formatted as a 2314, device 3333 is formatted as a 3330, and device 3344 is formatted as a 3340.) This service supports logical block sizes of 512, 1024, 2048, and 4096 bytes.

**Note:** The CMS 4K block structure on the first track of a 3340 disk is formatted differently than the other tracks of a 3340 CMS disk. The first track of the mini-disk contains three blocks. The first block has a length of 80 bytes, the second, 4096 bytes, and the third, 80 bytes. The remainder of the mini-disk is formatted as usual, two 4096-byte blocks on each track.

### Establishing Communications with DASD Block I/O Service

The CMS RESERVE command and the CMS DISKID function should be issued before using the DASD Block I/O System Service. These two facilities enable you to create a uniquely organized CMS file on a DASD and obtain information about the file needed to use the DASD Block I/O System Service. For further information, see "Using the DASD Block I/O System Service from CMS" in Part 2 of this manual, or see the *VM/SP CMS Command and Macro Reference*.

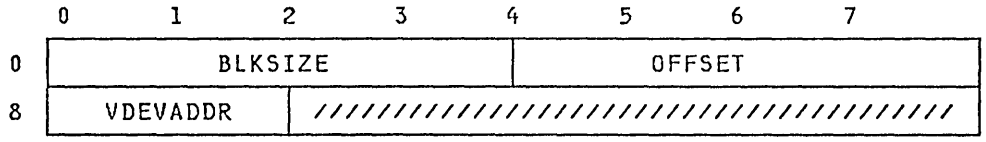
DASD Block I/O uses IUCV to set up communication between itself and a virtual machine. The IUCV macro checks the validity of all the IUCV parameters. Any IUCV errors are handled according to IUCV specifications. The DASD Block I/O System Service checks the validity of all the parameters it requires. Any errors resulting from this check are handled as described in the following sections.

IUCV requires that the virtual machine issues a DECLARE BUFFER command to initialize the virtual machine for IUCV communication. This command also specifies a buffer where IUCV can store external interrupt information. After communications is established with IUCV, the virtual machine must issue a CONNECT command to establish a path between itself and the target communicator. The target communicator, in this case, is the DASD Block I/O System Service. Only one CONNECT may be issued to userid *BLOCKIO for each virtual device that is intended to receive I/O requests.

No special authorization is required for a virtual machine to use DASD Block I/O. The MAXCONN (maximum connection) limit in the directory can be enlarged to satisfy the user's requirements. The DASD Block I/O System Service allows connections from any user.

#### *IUCV CONNECT to the DASD Block I/O System Service*

An IUCV CONNECT is issued by the virtual machine with USERID=*BLOCKIO and PRMDATA=YES specified in the IUCV CONNECT parameter list. In this case, IPUSER, the user data field in the IUCV parameter list, must have the following format. These values are obtained by the CMS DISKID function.




*where:*

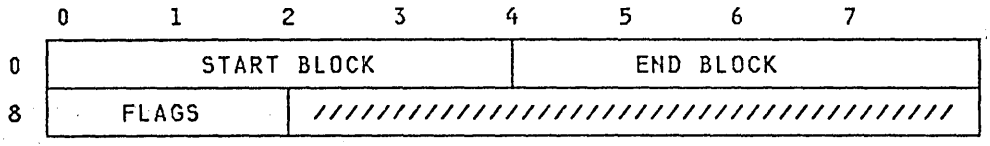
BLKSIZE — is the block size of the specified disk. The block size may be 512, 1K, 2K, or 4K bytes.

OFFSET — associates a physical block number to the first user data block of on the disk. Note that this number represents the number of sequential blocks used on the disk by the CMS file system to implement its structure. The DASD Block I/O System Service does not check the validity of this number. Therefore, the application may change this number if desired, but you could overlay files used by CMS.

VDEVADDR — is the virtual device address of the disk where the Block I/O is to be performed.

All reserved fields must be set to zero.

If all the parameters required by DASD Block I/O are valid, DASD Block I/O issues an IUCV ACCEPT on the path specifying PRMDATA=YES. The following information is returned in the IPUSER field of the IUCV Connection-Complete external interrupt buffer:




*where:*

START BLOCK — 1 minus the OFFSET specified on the IUCV CONNECT. This value along with END BLOCK is the range of block numbers allowable on the DASD Block I/O request.

END BLOCK — The number of blocks on the specified virtual device minus the OFFSET specified on the IUCV CONNECT. This value along with START BLOCK is the range of block numbers allowable on the DASD Block I/O request.

FLAGS — A set of bits defining the status of the virtual device. One bit is defined and the others are reserved.

| | | |
|---|---|---|
| RDONLY | X'0001' | Virtual device is read only |
| Unused | X'FFFE' | Zero |

All reserved fields are returned as zero.

If any of the parameters passed to DASD Block I/O are invalid, DASD Block I/O issues an IUCV SEVER on the path and flags the error. The first byte of the IPUSER field contains one of the following error codes:

X'01' Virtual device is not defined
X'02' Virtual device is not supported
X'03' Block size is not supported
X'04' IUCV path already exists for this device
X'05' Connection is not using PRMDATA=YES
X'06' Reserved field is not set to zero

### *IUCV SEND to the DASD Block I/O System Service*

When the connection to the device is ACCEPTed by DASD Block I/O, you can start sending I/O requests to DASD Block I/O. You can specify TRGCLS=, DATA=PRMMSG, and the PRMMSG= options on the IUCV SEND or you can move the necessary data into the IUCV parameter list yourself. The TRGCLS= option sets the type of I/O requested, read or write. The DATA=PRMMSG option sets a flag in IPFLAGS1, and the PRMMSG= option moves the block number and virtual buffer address into the IUCV parameter list. The following list defines the input necessary for the DASD Block I/O System Service on an IUCV SEND command:

IPRMMSG1 Block number
IPRMMSG2 Virtual buffer address
IPTRGCLS Block I/O service requested
F'01' Write request (CMS formatted)
F'02' Read request (CMS formatted)

If you have misused IUCV protocol set up for this system service, DASD Block I/O issues an IUCV SEVER on the path and flags the error. The first byte of the IPUSER field contains one of the following error codes:

X'07' IUCV communication was not sent using DATA=PRMMSG
X'08' No one-way messages are allowed on the path

Otherwise, DASD Block I/O tries to perform the request. It issues an IUCV REPLY to return the results of the I/O requests. One of the following return codes is returned in the IPRMMSG1 field of the IUCV parameter list:

F'00' I/O completed successful
F'01' Invalid block number
F'02' Invalid data buffer address
F'03' Write on read/only DASD
F'04' Incorrect block size - format error
F'05' Unrecoverable I/O error
F'06' Invalid service requested
F'07' Protection exception on virtual buffer

If the device is reset, the path is QUIESCEd and no more requests are allowed. When there are no I/O requests outstanding, DASD Block I/O issues an IUCV SEVER on the path and flags the error. The first byte of the IPUSER field contains the following return code:

X'09' Virtual device has been reset

When all communications with the DASD Block I/O System Service are completed, you can terminate communications by issuing either an IUCV SEVER or/and IUCV RETRIEVE BUFFER.

## The Special Message Facility

The Special Message Facility enables a virtual machine to send messages to another virtual machine by issuing the CP SMSG command. The Special Message Facility may be used with the Virtual Machine Communication Facility (VMCF) or with the Inter-User Communication Vehicle (IUCV). However, the sending virtual machine does not need to perform the initialization required by VMCF or IUCV. Initialization is handled by CP and is described later in this topic.

To send a message, a virtual machine need only prepare the message text -- up to 129 bytes -- and issue the class G command, SMSG. Parameters on the SMSG command identify the USERID of the receiving virtual machine and specify the message text. The format of the message text must be acceptable to the receiving virtual machine. The SMSG command is described in the *VM/SP CP Command Reference for General Users*.

Before the receiving virtual machine can receive special messages via VMCF, it must:

- Enable itself to receive external interrupts.
- Set bit 31 of control register 0 to a value of 1.
- Authorize itself by issuing DIAGNOSE Code X'68', AUTHORIZE. The parameter list, VMCPARM, specified with DIAGNOSE Code X'68' must contain a pointer to an external-interrupt buffer, must specify a buffer length of 169 bytes, and must have the special message flag (VMCPSMSG) turned on.
- Turn on this special message flag (VMCPSMSG) by setting VMCPSMSG to a value of B'1' or by issuing the class G command, SET SMSG ON. For information on using DIAGNOSE Code X'68', see "Description of VMCF Subfunctions" and "Invoking VMCF Subfunctions."

To understand how a special message is presented to the receiving virtual machine via VMCF, see "The SENDX Protocol" in the section "VMCF Protocol".

Before the receiving virtual machine can receive special messages via IUCV, it must do the following:

- Enable itself to receive external interrupts
- Set bit 30 of control register 0 to a value of 1
- Issue the IUCV DECLARE BUFFER function
- Issue the IUCV CONNECT function to the CP Message System Service
- Turn on the special message flag by issuing the class G command SET SMSG IUCV.

When a virtual machine no longer wishes to accept special messages, it may turn off the special message flag by issuing the command SET SMSG OFF. To resume receiving messages, the virtual machine may issue the command SET SMSG ON or

SET SMSG IUCV. CP sends an error message to any virtual machine that attempts to send a special message to another virtual machine that is not accepting special messages.

CP handles VMCF/IUCV initialization and special message processing as follows. When the SMSG command is issued, CP verifies that no invalid options were specified and that a valid USERID was specified. CP also verifies that the receiving virtual machine is accepting special messages. CP then obtains storage for the message, builds the appropriate parameter list, and sends the message to the receiving virtual machine.

# Single Console Image Facility

The Single Console Image Facility allows one user logged on to a single virtual machine to control multiple disconnected virtual machines. CP prefixes any output coming to the controlling virtual machine, from or on behalf of the originating virtual machine, with the userid of the originating virtual machine. The controlling virtual machine communicates with the virtual machines it is controlling via the CP class G SEND command.

The user whose virtual machine is being controlled is the primary user. The user whose virtual machine controls the primary user's virtual machine is the secondary user. The secondary user may run disconnected if he has a valid path to the IUCV Message System Service. Refer to "The Message System Service" in Part 1 of this publication for more information.

## Using the Single Console Image Facility

To enable a virtual machine to use the single console image facility, the installation must specify the userid of the secondary user on the CONSOLE directory control statement of the primary user. See *VM/SP Planning Guide and Reference* for a description of the CONSOLE directory control statement.

When the primary user disconnects his virtual machine and the secondary user is logged on, the secondary user receives control of the primary user's virtual machine. Even if the secondary user is not logged on when the primary user disconnects, the secondary user receives control of the disconnected virtual machine whenever he does logon. The primary user can regain control of his virtual machine at his own terminal by entering the LOGON command.

After the primary user disconnects, all console output from the disconnected virtual machine appears on the console of the secondary user if he is logged on. Output from the primary user's disconnected virtual machine is prefixed with the userid of the primary user.

The secondary user uses the CP SEND command to communicate with the primary user's disconnected virtual machine. See *VM/SP CP Command Reference for General Users* for a description of the SEND command.

**Notes:**

1. When the message, 'DMKQCO150A USER userid HAS ISSUED A CP READ' is received by the secondary user, the secondary user must reply with a SEND command, sending a CP command to the disconnected user named in the message.

2. When the message, 'DMKQCO150A USER userid HAS ISSUED A VM READ' is received by the secondary user, the secondary user must reply with a SEND command, sending a virtual machine command or a virtual machine reply to the disconnected user named in the message.

3. The console attributes of the secondary user are used for the display of messages. For example, if the primary user console is spooled TERM and the secondary user console is spooled NOTERM, only the messages that would normally be displayed with the NOTERM option are displayed at the secondary user's console.

# VM/SP Use of the IBM 3850 MSS

Virtual machines operating CMS, OS/VS1, or OS/VS2 (MVS) may access mass storage volumes containing VM/SP minidisks or entire mass storage volumes dedicated to the virtual machine. These volumes appear to the virtual machine as 3330 volumes and are accessed using 3330 device support in the virtual machine. VM/SP controls allocation, volume mounting, and volume demounting. Virtual machines that run OS/VS1 or OS/VS2 (MVS) with MSS support can also access mass storage volumes using dedicated device support.

## VM/SP Access to the MASS Storage Control

Whenever an MSS 3330V volume must be mounted or demounted, the VM/SP control program first selects an appropriate device address. If a volume mount is required, the device is selected from the pool of available 3330V devices created at system generation time. If a volume must be demounted, CP selects the address of the device on which the volume is currently mounted.

To pass mount and demount orders, the virtual machine must have an MSC port dedicated to it via the ATTACH command or the DEDICATE directory statement. An application program named DMKMSS is distributed as part of VM/SP; it acts as an interface between CP and the MSC. After DMKMSS is started in an OS/VS1 or OS/VS2 (MVS) virtual machine, it uses a special virtual I/O device and the VM/SP DIAGNOSE interface to communicate with the VM/SP control program.

If the MSC request was for a volume mount, the MSC ending status indicated that the MSC was processed. If the MSC accepts the mount order, the MSC orders the staging adapter to generate a pack change interrupt (an unsolicited device end) on the device when that device has been mounted. CP receives the pack change interrupt, the RDEVBLOK is set to indicate that the volume is mounted, and any VM/SP task waiting for the volume is marked dispatchable. If the mount order was rejected, no further processing of the mount occurs. The previously allocated RDEVBLOK is marked free and processing continues with the next MSS request.

## Asynchronous MSS Mount Processing

When an MSS volume mount is required to satisfy a LINK or ATTACH command or an MDISK or DED directory statement, CP returns control to the virtual machine as soon as MSC accepts the mount request[3]. The virtual machine may continue to execute before the virtual device specified on the MDISK, DED, LINK, or ATTACH is available.

The reasons for asynchronous MSS mount processing are the relatively long time required to complete the mount, and the chance that an error may occur in the MSS after the mount order is accepted. The virtual device to be mounted may not be vital to the specific task to be accomplished. Also, if an error occurs in the MSS (such as a permanent read error on a cartridge) after the mount is accepted, the error indication is passed from the MSC to the virtual machine. VM/SP cannot

[3] However, the central server cannot issue these CP commands. The central server is the MSS communicator virtual machine which acts as an interface between CP and the MSC. CP commands issued to the central server are ignored and a message is issued.

determine that an error has occurred and that the mount will not complete. If the virtual machine were not dispatchable until the mount completed, it would be locked out until the MSS error was corrected.

With asynchronous mount processing, the virtual machine has the flexibility to either continue processing without the affected virtual device, or wait until the MSS mount completes. If the virtual machine issues an SIO instruction to a virtual device that is defined on the volume being mounted, VM/SP queues the I/O request until the mount completes. The virtual machine is marked I/O wait nondispatchable until the mount completes and the SIO is started.

## VM/SP Processing of MSS Cylinder Faults

VM/SP supports 3330V cylinder fault processing in two ways: real channel programs directed to 3330Vs are constructed so that cylinder faults can be recognized and the channel program restarted; and the attention interruption (indicating that the cylinder fault has been satisfied) is recognized and any I/O for that device restarted.

When the VM/SP processor issues a seek CCW to a 3330V device, the staging adapter must translate the seek argument to the correct cylinder of staging space. If the cylinder referenced in the seek is staged, then the SIO is passed to the associated staging DASD drive. If the referenced cylinder is not staged, the staging adapter initiates cylinder fault processing. The staging adapter first passes a cylinder fault indication to the MSC, requesting the cylinder of data to be staged. It then returns a status modifier to the channel in response to the seek, which causes the channel to skip one CCW in its CCW fetch processing. That is, the channel does not fetch the next CCW after the seek.

As a result of the cylinder fault, the MSC allocates staging space for the requested data and causes it to be staged. The staging adapter then generates a channel end/device end interruption to indicate that the cylinder has been staged.

It is possible in error situations that the attention interruption may not be received. Each time an I/O request is queued by VM/SP as a result of a cylinder fault, a timer is set. If the timer expires before the interruption is received, a message (DMKSSS074I) is written to the VM/SP system operator and the request is retried.

## Backup and Recovery of MSS Volumes

The process of creating backup copies of MSS volumes, and restoring from those backup copies, can be controlled through the OS/VS access methods services COPYV command. This command can operate without system operator intervention.

For each active volume in the MSS, there may be one or more copy volumes. At any time, the active volume may be copied to a copy volume with the access method services COPYV command. All volume mounts and data transfer are controlled by this command. If at any time it is necessary to restore the level of a volume to that of a copy, the OS/VS access methods services RECOVERV command is used.

All the OS/VS access methods services commands can be run from either a real processor or a VS virtual machine. If the MSS communicator virtual machine is in operation, these commands can be run from that virtual machine while it is acting as the communicator.

# Logical Device Support Facility

The Logical Device Support Facility allows an application running in a virtual machine to create within CP one or more non-extended (i.e. no extended color, extended highlighting, or programmed symbols) logical 3270 display devices. Except for the logical device facility, CP is unaware of the fact that this terminal has no real existence and is driven by the application program. In particular, the CP display terminal support sees it as a local 3270. Any output directed to a logical device is redirected to the virtual machine for which the device was created. The virtual machine can also transfer data to CP to be entered as input from a specific logical device, as if it were interactively produced on a real terminal.

The Logical Device Support Facility is made up of two data transfer subfunctions, three control subfunctions, a special external interrupt (code X'2402'), and an external control word for passing control information with the external interrupt.

To implement this facility, subfunctions are invoked using the DIAGNOSE instruction (code X'7C'). Registers Rx, Rx+1, Ry, and Ry+1 are used to indicate the subfunction, logical device identification, and other subfunction-dependent information.

A special interrupt code (X'2402') is used by module DMKHPS to notify a virtual machine of pending logical device status for a logical device created for that virtual machine. Along with this interrupt, the virtual machine receives a control word at a virtual storage location indicating the ID of the associated logical device and the reason for the interrupt.

Figure 27 on page 204 is a summary of Logical Device Support Facility subfunctions. More complete information about each of these subfunctions is included under "Descriptions of Logical Device Support Facility Subfunctions."

Data is directed to a logical device using the logical device ID. This ID is assigned by CP during execution of the INITIATE subfunction. Data transfer takes place within CP at a channel command level. I/O directed to a logical device proceeds within CP via the normal path for a local 3270 terminal up to the point that DMKIOS is normally called to start I/O. At that point, control passes to DMKHPS to process the CCW string. Channel commands requiring interaction cause external interrupts to the virtual machine for which the associated logical device was created.

The format of data from the virtual machine must conform to 3270 architecture for local display stations. Extended data streams are not supported.

Up to eight virtual machines may simultaneously create logical devices, and each virtual machine can create up to 512 of these devices.

| Subfunction | Description |
|---|---|
| INITIATE | Initiate logical device communications |
| ACCEPT | Transfer data written to logical device to virtual machine storage. |
| PRESENT | Transfer data from virtual machine to CP as input from logical device. |
| TERMINATE | Drop a specific logical device. |
| TERMINATE ALL | Drop all logical devices created for this virtual machine. |

**Figure 27. Summary of Logical Device Support Facility Subfunctions**

VM/Pass-Through Facility program product is an example of an application using the Logical Device Support Facility. Through the combined support of these two facilities, a VM/SP user attached to system A via a 3270 Display Station can access VM/SP system B as though the display station were locally attached to system B.

# Timers in a Virtual Machine

This section describes the results obtained in using timers in a virtual machine created by CP.

## Interval Timer

Virtual location 80 (X'50'), the interval timer, contains different values than would be expected when operating in a real machine. On a real machine, the interval timer is updated 300 times per second when enabled and when the real machine is not in manual state. The interval timer on a real machine thus reflects system time and wait state time. In a virtual machine, the interval timer reflects only virtual processor time, and not wait time. It is updated by CP whenever a virtual machine passes control to CP, and this one updating reflects the entire time the virtual machine had control. Note that during the time a virtual machine has control, the virtual interval timer does not change; the virtual processor time used is added to the virtual interval timer when CP regains control. For some privileged instructions, CP may be able to simulate the instruction and still return control to the virtual machine before the end of that virtual machine's time slice. In such cases, the virtual interval timer is updated but only for those privileged instructions that require normal or fast reflect entry into the dispatcher. For those privileged instructions that do not require entry into the dispatcher, the virtual interval timer is not updated until CP gets control at the end of the time slice.

If the virtual machine assist feature or Extended Control - Program Support is ON, more time is charged to the virtual interval timer than if the feature is OFF. When the virtual machine assist feature is OFF, the time spent by CP to simulate privileged instructions *is not* charged to the virtual interval timer; whereas, with the feature ON, the time spent *is* charged to the virtual interval timer.

### Virtual Interval Timer Assist

The virtual interval timer assist feature is the updating of the virtual interval timer and presentation of timer interrupts to the virtual machine by the hardware. When the software simulates the interval timer, updating occurs only when CP takes over control. This usually results in an update frequency of once per time slice and repeatability of timed intervals suffers greatly under these conditions. When the virtual interval timer assist feature is active, the update frequency is the same for both virtual and real interval timers, 300 times a second.

In order for the virtual interval timer assist feature to be active, the following conditions must be met:

- VM/SP must be running on a Model 135-3, 138, 145-3, 148, 3031, 3031AP, 4331, or 4341.
- The virtual machine must have enabled the virtual machine assist and the virtual interval timer (SET TIMER {ON | REAL}).
- The virtual machine must have enabled both the virtual machine assist and the virtual interval timer assist (SET ASSIST ON TMR).

VM/SP provides an option, called the REALTIMER option, which causes the virtual interval timer to be updated during virtual wait state as well. With the REALTIMER option in effect, a virtual interval timer reflects virtual processor time and virtual wait time, but not CP time used for services for that virtual

machine, such as privileged instruction execution. The more services a virtual machine requires from CP, the greater the difference between the time represented by the interval timer and the actual time used by and for the virtual machine. The larger the number of active virtual machines contending for system resources, the greater the difference between virtual machine time and actual elapsed (wall clock) time.

## Processor Timer

A virtual machine must have the ECMODE directory option to use the System/370 processor timer.

The processor timer is supported in a virtual machine in much the same way as is the interval timer. That is, the processor timer in a virtual machine records only virtual processor time, and it is updated when the virtual machine passes control back to CP.

If the real timer option is specified, the virtual processor timer reflects all actual elapsed time except CP time used for services, such as privileged instruction execution, for that virtual machine.

The method of sampling the value in the virtual processor timer causes it to appear to a virtual machine to be updated more often than an interval timer. The privileged instructions Set processor Timer (SPT) and Store processor Timer (STPT) are used to set a doubleword value in the virtual processor timer and to store it in a doubleword location of virtual storage. When a virtual machine samples the value in the virtual processor timer by issuing a STPT instruction, CP regains control to execute the privileged instruction, and updates the time. The act of sampling the processor timer from a virtual machine causes it to be brought up to date.

## TOD Clock

The System/370 time-of-day (TOD) clock does not require simulation in a virtual machine. The System/370 in which CP is operating may have one real TOD clock for each processor, and all virtual machines can interrogate the real TOD clock. The Store Clock (STCK) instruction is nonprivileged; any virtual machine can execute it to store the current value of the TOD clock in its virtual storage. The Set Clock (SCK) instruction, which is used to set the TOD Clock value, can be issued from a virtual machine, but CP always returns a condition code of zero and does not actually set the clock. Note that the TOD clock is the only true source of actual elapsed time information for a virtual machine. The base value for the TOD clock in VM/SP is 00:00:00 GMT, January 1, 1900.

In an attached processor or multiprocessor environment, the TOD clocks are synchronized using the procedure described in the *IBM System/370: Principles of Operation*.

## Clock Comparator

The clock comparator associated with the TOD clock is used in virtual machines for generating interrupts based on actual elapsed time. The ECMODE option must be specified for a virtual machine to use the clock comparator feature. The Set Clock Comparator (SCKC) instruction specifies a doubleword value that is placed in the clock comparator. When the TOD clock passes that value, an interrupt is generated.

## Pseudo Timer

The pseudo timer is a special VM/SP timing facility. It provides 24 or 32 bytes of time and date information in the format shown in Figure 28.

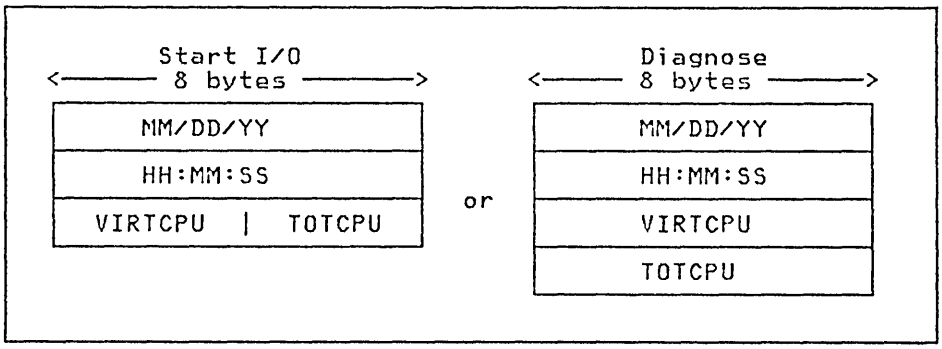



**Figure 28. Formats of Pseudo Timer Information**

The first eight-byte field is the date, in EBCDIC, in the form Month/Day-of-Month/Year. The next eight-byte field is the Time of Day in Hours:Minutes:Seconds. The VIRTCPU and TOTCPU fields contain virtual processor and total processor time used. The units in which the processor times are expressed and the length of the fields depend upon which of two methods is used for interrogating the pseudo timer.

### *Pseudo Timer Start I/O*

The pseudo timer can be interrogated by issuing a START I/O to the pseudo timer device, which is device type TIMER, and is usually at device address 0FF. No I/O interrupt is returned from the SIO. The address in virtual storage where the timer information is to be placed is specified in the data address portion of the CCW associated with the SIO. This address must not cross a page boundary in the user's address space. If this method is used, the virtual processor and the total processor times are expressed as fullwords in high resolution interval timer units. One unit is 13 microseconds.

### *Pseudo Timer DIAGNOSE*

The pseudo timer can also be interrogated by issuing DIAGNOSE with an operation code of C, as described under "DIAGNOSE Instruction in a Virtual Machine." If this method is used, the virtual and total processor times are expressed as doublewords in microseconds.

# CP in Attached Processor and Multiprocessor Modes

This chapter enables you to:

- Define attached processor (AP) mode
- Define multiprocessor (MP) mode
- Understand the use of the channel set switching instructions when available
- Understand the use of the privileged instructions that set and inspect the processor's prefix register
- Understand the use of the privileged instruction that determines the address of the processor that is executing
- Understand the use of hardware signaling to communicate between processors
- Understand the use of a TOD clock synchronization check
- Code fetch and store sequences that can be safely used in the AP/MP environment
- Use locks for serialization of functions
- Set processor affinity
- Change processors using the SWTCHVM macro
- Configure I/O devices to obtain maximum availability and recovery potential
- Debug an AP/MP system.

## Multiprocessor Environment

In a tightly coupled multiprocessor (MP) environment two processors share real storage under the control of a single control program. Both processors have I/O capability in an MP environment. See the section "Configuring I/O Devices" for a discussion on how to configure I/O devices for maximum availability and recovery potential.

In a dyadic environment two processors share real storage under the control of a single control program. Both processors have I/O capability. However, unlike an MP complex, a dyadic processor cannot be partitioned into two distinct uniprocessor systems.

## Attached Processor Environment

In an attached processor (AP) environment two processors share real storage under the control of a single control program. However, unlike a multiprocessing environment, only one processor in an AP environment has I/O capability. If you are running on a 3033 or a 3081, the channel set switching feature is available. If a severe hardware error occurs on the first processor in an AP environment, the control program may be able to use the channel set switching feature to dynamically

switch the channels of one processor to the other processor. The channel set switching instructions that the control program can use to connect and disconnect a channel set to a processor are:

```
CONCS   connect channel set
DISCS   disconnect channel set
```

**Note:** When you generate VM/SP as an MP system it does not use the channel set switching facility even if the facility is installed on the hardware.

## Advantages of the AP/MP Environment

An AP/MP environment provides additional processing capability when compared to a uniprocessor environment. An AP/MP environment also provides increased availability. In case of hardware malfunction on one processor, the other processor can frequently continue operating. Serviceability is enhanced because it is possible to use the VARY ON/OFF PROCESSOR command to vary a processor offline for system repair or to upgrade the system.

## Facilitating an AP/MP Environment

In an AP or MP environment, two processors share main storage. To facilitate this sharing, VM/SP provides for the unique features and requirements of this environment: prefixing, processor address identification, processor signaling, time-of-day clock synchronization, interlocks on certain fetch and store instructions, locks, and affinity setting. The system programmer should be familiar with the instructions used to accomplish these tasks.

### *Prefixing*

When VM/SP is executing in an AP/MP environment both processors cannot use absolute page zero for status information. Instead, each processor has its own prefixed storage area (PSA) in the high end of real storage. However, if the system operator varies a processor online after CP initialization completes, the processor's PSA may be located in any page of the dynamic paging area. See Figure 29 on page 210 for a storage map of the V=R machine after CP initialization.

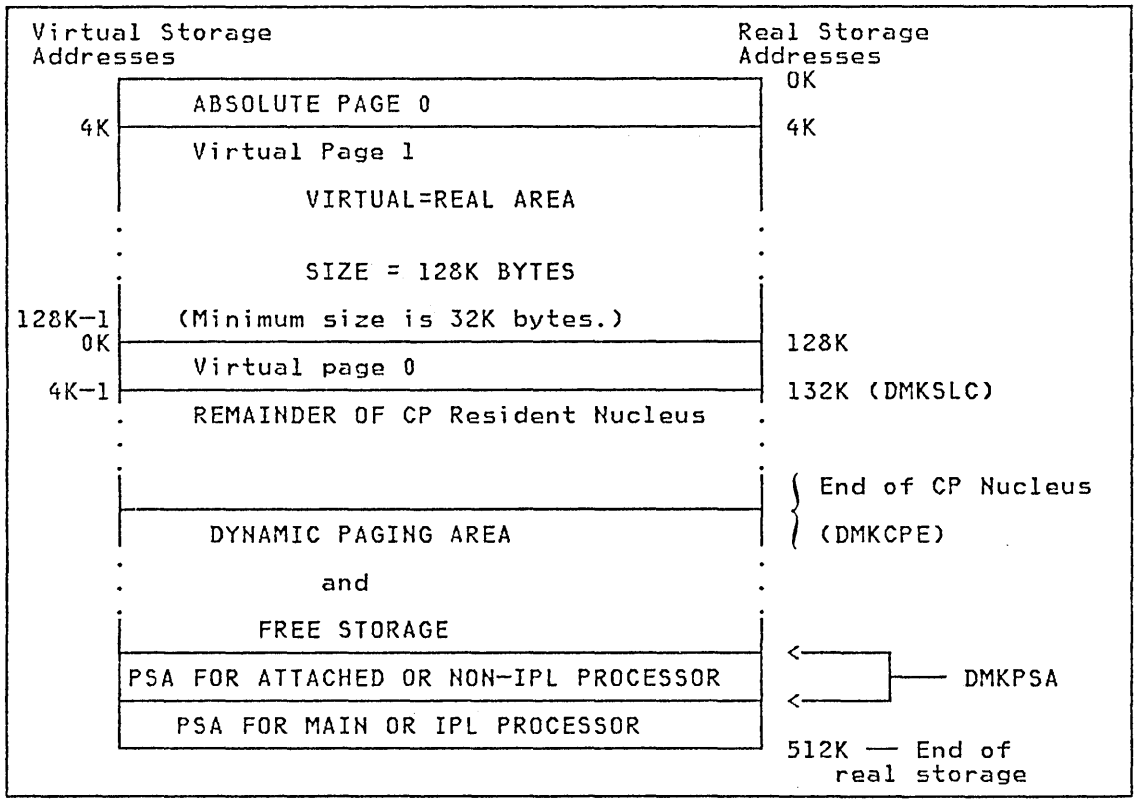



**Figure 29. Storage Layout in a Virtual=Real Machine**

### Prefix Registers

The control program puts the addresses of the PSAs in the prefix registers of the two processors during system initialization. The control program can set and inspect the contents of the processor's prefix register by using the privileged instructions:

- SPX - set prefix
- STPX - store prefix

If you are operating in AP/MP mode, VM/SP uses the prefix registers. When code executing on either processor references an address from 0 to 4095, the referenced address is added to the contents of the prefix register for that processor to produce the absolute address that will be accessed. In this way, each processor can independently control its operations with separate channel address words and channel status words. Prefixing is described in detail in *System/370 Principles of Operation*.

## *Identifying a Processor Address*

The hardware assigns the processor address during system installation. To determine the address of the processor that is executing, the control program issues the privileged instruction:

- STAP store CPU address

VM/SP stores both processor addresses in both PSAs in the following fields:

- IPUADDR CPU address of this processor
- IPUADDRX CPU address of the other processor

The system uses this information for interprocessor communication.

## *Signaling*

During certain critical periods, such as when a processor malfunctions or when a processor synchronization must occurs, one processor must signal the other processor. There are three types of program-controlled signals possible under VM/SP. They are:

- Emergency signals
- Direct signals
- External call signals

### SIGNAL Macro

Use the SIGNAL macro to issue the signal processor (SIGP) instruction. If you have generated the system as an AP/MP system, the control program expands the macro. The macro expansion code destroys the contents of registers 0, 1, 14, and 15. The macro expansion loads register 0 with the signalled processor address, loads register 1 with the function code, and uses registers 14 and 15 for linkage.

**Note:** If you have not generated the system as an AP/MP system, the control program treats the SIGNAL macro as a no-operation.

The SIGNAL macro causes all signaling requests to be sent to the external interruption handler so that error analysis and recovery attempts are centralized.

The format of the SIGNAL macro and the functions that you can perform using each type of signal are:

| | | | |
|---|---|---|---|
| label | SIGNAL | CLKCHK<br>EXTEND<br>QUIESCE<br>SHUTDOWN<br>SYNC<br>XTNDEXIT | [ ,CONTROL=SERIAL] |
| | | APR<br>DISPATCH<br>RESUME<br>WAKEUP | [ ,CONTROL=[PARALLEL]<br>=[AUTO] ] |
| | | RESTART<br>START<br>STOP<br>SSS | [ ,CONTROL=[PARALLEL]<br>=[AUTO] ] |

where:

label
is any desired label.

is the function to be performed and is a required positional parameter. This parameter can be an emergency signal, an external call signal, or a direct signal.

*Emergency Signals*

When one processor wants the other processor to perform an action immediately, it executes an emergency signal instruction. Since emergency signals can only be serial, control is not returned to the issuing processor until the other processor performs the function. The emergency signals are:

CLKCHK - indicates that the high order bits of the time-of-day clocks are not synchronized

EXTEND - indicates that free storage extend processing is to take place

QUIESCE - indicates that the receiving processor is to halt all execution until a RESUME signal is received

SHUTDOWN - indicates that the system is about to shutdown

SYNC - indicates that the low order bits of the time of day clocks are no longer synchronized

XTNDEXIT - indicates that free storage extend process is complete and virtual machines can be dispatched again

*External Call Signals*

When one processor wants to call the other processor's attention to an event or condition, it executes an external call order. The external call functions are:

APR - causes automatic processor recovery to be invoked to attempt to remove a failing processor from the configuration

DISPATCH - indicates that a CPEXBLOX is on the dispatcher's queue for the receiving processor

RESUME - cancels a previous QUIESCE signal

WAKEUP - indicates that the processor is to resume operations after having stopped processing

*Direct Signals*

Direct signals correspond to physical buttons on the real processor. These signals are controlled by the hardware, and cannot be masked off. The direct signals are: RESTART, START, STOP, and SSS (stop and store status).

CONTROL=
is the second operand.

CONTROL=SERIAL - specifies that control returns to the sender after the function is complete. CONTROL=SERIAL is the only parameter that you can use with the emergency signals. You cannot specify CONTROL=SERIAL for the external calls and direct signals.

CONTROL=SERIAL - specifies that control returns to the sender after the function is complete. CONTROL=SERIAL is the only parameter that you can use with the emergency signals. You cannot specify CONTROL=SERIAL for the external calls and direct signals.

CONTROL=PARALLEL - specifies that control returns to the sender even though the function may not be complete. You can use CONTROL=PARALLEL with the external calls and direct signals; it is the default for these signals.

CONTROL=AUTO - specifies that the signal is sent to the issuing processor. You can use CONTROL=AUTO with the external calls and direct signals.

### *Time-of-Day (TOD) Clock Synchronization Check*

If more than one TOD clock exists in a tightly-coupled configuration, the clocks must be synchronized. If the time-of-day (TOD) clocks are not in high order synchronization during system initialization of an AP/MP system, the system issues a message to the system operator to enable the TOD clock set key. If the clocks are out of low order synchronization, that is bits 32 to 63 of the two clocks do not match, the system receives a time-of-day-clock-sync-check when external interruptions are enabled. Then the system synchronizes the clocks.

### *Fetching and Storing*

Since main storage is shared, there is a possibility that both processors may be accessing the same location in storage simultaneously. The control program must prevent simultaneous updates to the same storage location. In a tightly-coupled multiprocessor environment certain instructions cannot safely execute if there is a chance that their execution might change storage that the other processor is also using. Fetch and store instructions such as OI, NI, and NC could cause one processor to update storage that the other processor is also using. To prevent this type of error in a multiprocessing environment, the following fetch and store instructions have interlocks:

- CDS - compare double and swap
- CS - compare and swap
- TS - test and set

The following example shows how you could use the compare and swap instruction to set a flag in a multiprocessing environment.

```
        Processor A

        LA   Rx,FLAGS      load Rx with address of FLAGS byte
        LA   Ry,X'80'      load Ry with byte to set FLAGS
        SLL  Ry,24         line up fields
        L    Rz,0(Rx)      load Rz with FLAGS byte
RETRY   LR   Rw,Rz         load Rw with contents of Rz
        OR   Rw,Ry         load Rw with reset value of FLAGS
        CS   Rz,Rw,0(Rx)   reset FLAGS byte if =; otherwise load Rz from FLAGS
        BNE  RETRY         if contents of Rz ≠ FLAGS, branch to RETRY
        .
        .
        .
FLAGS   DC   X'20'         initial setting of field



        Processor B

        LA   Ra,FLAGS      load Ra with address of FLAGS byte
        LA   Rb,X'40'      load Rb with byte to set FLAGS
        SLL  Rb,24         line up fields
        L    Rc,0(Ra)      load Rc with FLAGS byte
RETRY   LR   Rd,Rc         load Rd with contents of Rc
        OR   Rd,Rb         load Ra with reset value of FLAGS
        CS   Rc,Rd,0(Ra)   reset FLAGS byte if =; otherwise load Rc from FLAGS
        BNE  RETRY         if contents of Rc ≠ FLAGS, branch to RETRY
        .
        .
        .
FLAGS   DC   X'20'
```

**Figure 30. Sample of the Correct Way to Set a Flag in an AP/MP Environment**

## *Locks and Serialization of Functions*

If VM/SP is executing in AP/MP mode, critical sections of code must be serialized. A critical section of code is code that is executing on one processor and must appear as one indivisible operation to the other processor. An example of a critical section of code would be code that updates a queue. The other processor should not have access to the queue until the element is either added or deleted and all pointers are updated. VM/SP uses locks to accomplish serialization of critical functions. A lock is an area of storage. It is initialized to a value, usually zero, to signify that the lock is not held. Before entering a critical section of code, the processor requests the lock to serialize the operation. The operating system determines if a lock is free and gives it to the processor requesting the lock by means of a hardware interlocked update operation such as compare and swap (CS). When exit is made from the critical section of code, the system *releases* the lock by changing its value back to zero.

### Locking Hierarchy

The introduction of a locking structure makes the avoidance of processor deadlock a prime concern. A deadlock occurs if both processors each have a different lock and want to obtain the lock that the other processor holds. VM/SP uses a locking hierarchy to avoid these deadlock situations. A locking hierarchy provides for the ordering of the set of locks. If a processor holds a given lock, it can only request a

lock that is lower in the locking hierarchy. For example if a processor holds the free storage lock, the processor cannot perform input/output. On the other hand, if a processor holds the I/O lock, the processor can obtain free storage.

Figure 31 shows the hierarchy of locks under VM/SP where the global system lock is the highest lock. The real storage management lock and the I/O lock are on the same level. There are no situations which require simultaneous ownership of the I/O lock and the real storage management lock. If such a need arises, the system will define a hierarchy between these locks.

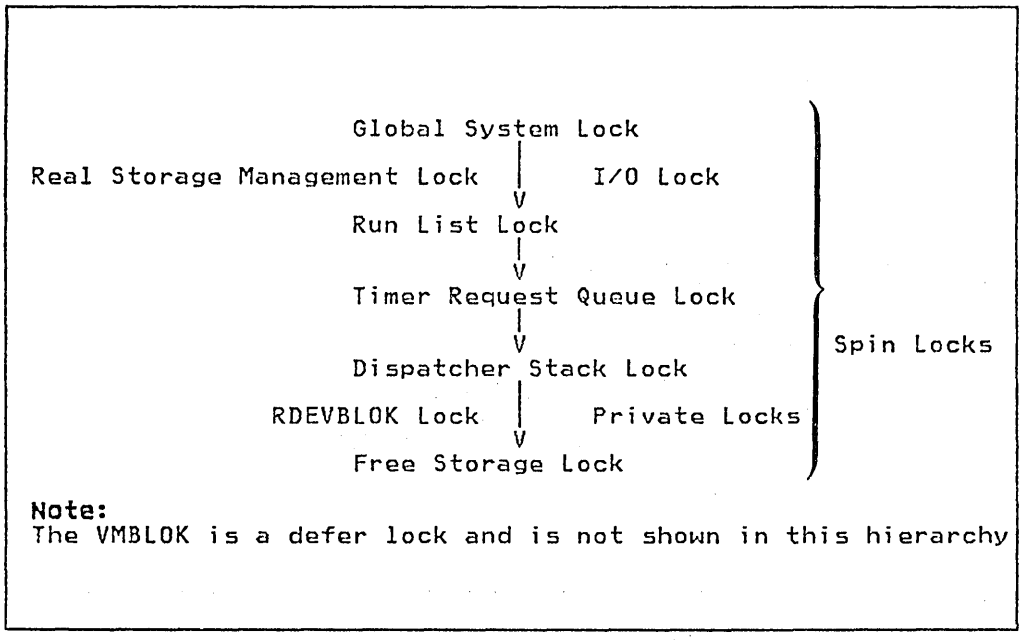



**Figure 31. Hierarchy of VM/SP Locks**

## Types of Locks

There are two basic types of locks:

- Defer locks
- Spin locks

If a function requests a defer lock and it is not available, control is returned to the caller with a condition code that indicates that the lock is not available. However, if a function requests a spin lock and it is not available, the lock manager loops until the lock becomes available.

## Locking Structure

To provide system integrity, VM/SP attached processor and multiprocessor support is designed around one global lock, a VMBLOK local lock, and several system local locks for specifically identified queues or modules.

***Global System Lock***: Much of CP runs under the global system lock, which is a defer lock. For example, all command processing requires the global system lock. Also, all code executed via an IOBLOK, TRQBLOK, or CPEXBLOK is protected

by the global system lock. Certain basic system functions, however, are able to execute without the global system lock on the mainline, non-error paths. These functions include virtual page fault processing, the simulation of virtual I/O requests and some other privileged operations, and the processing of a real I/O interruption.

If a processor needs the global system lock and cannot obtain it, the processor must defer the function until the global system lock is available. The function is deferred by either stacking the VMBLOK appendage (called the deferred interrupt block) or a CPEXBLOK for later processing. The processor that could not obtain the global system lock then uses the unlocked dispatcher entry to dispatch a new virtual machine.

In some situations, a function cannot be deferred even though the global system lock is not available. In these cases, the dispatcher spins on the global system lock until it becomes available. The dispatcher requires the system lock to unstack CPEXBLOKs, IOBLOKs, and TRQBLOKs.

To ensure system integrity along the paths that do not require the global system lock, other local locks have been defined. With the exception of the VMBLOK lock, these locks are all spin locks and are held for relatively short periods of time.

***VMBLOK Lock***: The VMBLOK lock, which is a defer lock, is obtained by the dispatcher before dispatching a virtual machine in problem program state or before performing any system service for that virtual machine. This lock prevents a virtual machine from being serviced by CP while it is running in problem program state.

***Real Storage Management Lock (RM Lock)***: The real storage management lock (called the RM lock) is a spin lock that serializes functions within the paging subsystem. This lock controls all accesses to the free and flush lists, the page read and write request queues, the deferred allocation queue, the active paging queue, CPEXBLOKs chained via CPEXMISC, and certain nonreentrant fields within DMKPTR and DMKPAG.

***I/O Lock***: The I/O lock is a spin lock that serializes access to I/O devices by serializing access to fields in the real I/O control blocks: RCHBLOK, RCUBLOK, and RDEVBLOK.

***Run List Lock***: The run list lock is a spin lock that controls all additions to and deletions from the run list.

***Timer Request Queue Lock***: The timer request queue lock is a spin lock that allows the external first-level interruption handler to process a timer interruption without the global system lock.

***Dispatcher Stack Lock***: The dispatcher stack lock is a spin lock that controls all additions to or deletions from the IOBLOK/TRQBLOK queue or the CPEXBLOK queue.

***RDEVBLOK Lock***: The RDEVBLOK lock is a private spin lock that the paging subsystem uses to serialize the IOBLOK queue.

***Free Storage Lock***: The free storage lock is a spin lock obtained by DMKFRE for all FREE and FRET requests for free storage. All of the locks that CP uses are described in further detail in *VM/SP System Logic and Problem Determination Guide.*

## LOCK Macro

Use the LOCK macro to obtain or release a lock. The format of the LOCK macro is:

```
label   LOCK   {OBTAIN },TYPE=(SYS     )  [,SPIN= {YES}] [,SAVE]
               {RELEASE}      (VMBLOK  )         {NO }
                              (FREE    )  [,OPTion=NOUPDT]
                              (RL      )
                              (TR      )
                              (DS      )
                              (IO      )
                              (RM      )
                              (PRIVATE )
```

where:

label
: is any desired label

OBTAIN

RELEASE
: is a required positional operand indicating whether the lock is to be obtained or released.

TYPE=
: is a required positional operand. The possible parameters are:

  SYS — for the global system lock

  VMBLOK — for the VMBLOK

  FREE — for the free storage lock

  RL — for the runlist lock

  TR — for the timer request queue lock

  DS — for the dispatch lock

  IO — for the I/O lock

  RM — for the real storage management lock

  PRIVATE — for a private user-defined lock If you have user-defined areas that are used by more than one virtual machine, you will need to define your own locking conventions. You can use the LOCK macro to obtain and release a private lock.

  The system programmer must specify the address of the lockword in register 1 and the lockword must be a fullword aligned on a fullword boundary. Spin time for private locks is kept in the DMKLOKSI timer value for all non-DMKLOK locks.

SPIN={YES | NO}
specifies whether control is to be returned without the lock being held. The default is SPIN=YES.

SAVE
is an optional keyword that indicates register 0, 1, 14, and 15 are to be saved before the rest of the macro expansion. These registers are saved in the PSA of the processor that is executing this macro. The registers are restored before exit from the macro expansion.

OPTION=NOUPDT
indicates that the VMBLOK should be locked without checking for shared segments.

*Condition Codes*

The condition code (cc) is set as a result of the invocation of the LOCK macro.

| Condition Code | Parameter | Meaning |
|---|---|---|
| cc=0 | OBTAIN | lock obtained |
| | RELEASE | lock released |
| cc=1 | OBTAIN,SPIN=NO | lock owned by another processor |

For various abend codes related to lock usage, see *VM/SP System Messages and Codes*.

## *Affinity*

When you specify the affinity option for a virtual machine, the program of that virtual machine is executed only on the specified processor. You might want to specify affinity in the following cases:

- If one processor has a special hardware feature or a special RPQ that is required for a particular program, set affinity to this processor.
- If a virtual machine has a high I/O-to-compute ratio, you might want to set affinity to the I/O processor. On the other hand, if a virtual machine has a high compute-to-I/O ratio, you could set affinity to the attached processor.

### How to Set Affinity

You request affinity either in the directory or with a SET AFFINITY command. See the *VM/SP CP Command Reference for General Users* for details on the class G SET AFFINITY command. See the *VM/SP Operator's Guide* for other privilege classes of the SET AFFINITY command.

## *Shared Segments in an AP/MP Environment*

When two processors are executing simultaneously, it is necessary to know when a user changes a shared page. In attached processor or multiprocessor mode, there are two sets of page tables and swap tables maintained for each shared segment unless a user is running unprotected. If a user is running unprotected shared segments, there is only one copy.

### *SWTCHVM Macro*

Routines that must lock a virtual machine other than the current virtual machine use the SWTCHVM macro. The SWTCHVM macro unlocks the VMBLOK specified in register 11 and locks the VMBLOK specified in register 1. Time charging is also switched. The format of the SWTCHVM macro is:

| | | |
|---|---|---|
| label | SWTCHVM | OPT={[STAY] [NOUPDT]} {UNLOCK} |

where:

label is any desired label

STAY indicates that if the VMBLOK lock is not available, a CPEXBLOK will be stacked for the current processor.

NOUPDT indicates that the VMBLOK should be locked without checking for shared segments.

UNLOCK indicates that the current VMBLOK is unlocked, register 11 is updated to point to VMBLOK specified in register 1, the timer is switched to start charging supervisor time to the new VMBLOK, but the new VMBLOK is not locked. Note: The UNLOCK option cannot be specified with either of the other options.

## Configuring and Debugging MP Systems

The user should keep the following things in mind when configuring I/O devices for an MP system and when debugging an AP/MP System.

### *Configuring I/O Devices for an MP System*

When you configure I/O devices, you should consider the following:

- The possibility of a hardware failure
- Smooth transition when you reconfigure between MP and uniprocessor (UP) modes for maintenance.

In either of these cases to ensure maximum system availability, you should provide paths from both processors to I/O devices. You can do this in several ways:

- Configure symmetrically as many channels and I/O devices as possible.
- Install channel-switching and string-switching features on control units where possible. A channel switch is a feature on a control unit that enables two real processors to share a symmetric device. A symmetric device is a device that can be accessed by both processors, while an asymmetric device cannot be shared. A string switch enables you to attach a symmetric I/O device to two separate control units. These features provide access to I/O devices from both processors. This increased access reduces the possible loss of access to critical I/O devices because of hardware malfunctioning.

- Configure asymmetric devices through a manual switching unit. Then the operator can physically attach these devices to either processor, one processor at a time. Asymmetric devices include printers, card readers, punches, and information display systems.

- Provide redundant control units for critical I/O devices.

## *Debugging an AP/MP System*

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

### PSA

A dump for a program operating in AP or MP mode contains three PSAs -- the absolute PSA, one for the IPL processor and one 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. See *VM/SP Data Areas and Control Block Logic, Volume 1* for an 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; while 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 the "CP Internal Trace Table" section and Figure 67 on page 499 earlier in this publication. Trace table pointers (the address of trace table start, the address of trace table end, and the address of the next available trace entry) 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 four fullwords of information. The first fullword contains the logical processor address of the owning processor. This will be zero if the lock is

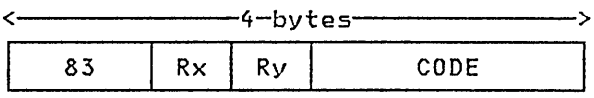
not held. The second fullword contains the value in the lock owner's register 12. The third and fourth fullwords contain the total amount of time spent spinning on this lock and the total number of spins respectively.

The VMBLOK lock is located in the VMBLOK at VMLOCK. 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. When the lock is held, RDEVIOBL contains the logical processor address of the owning processor.

## DIAGNOSE Instruction in a Virtual Machine

The DIAGNOSE instruction cannot be used in a virtual machine for its normal function. If a virtual machine attempts to execute a DIAGNOSE instruction, a program interrupt returns control to CP. Since a DIAGNOSE instruction issued in a virtual machine results only in returning control to CP and not in performing normal DIAGNOSE functions, the instruction is used for communication between a virtual machine and CP. The machine language format of DIAGNOSE is:




*where:*

83 is X'83' and interpreted by the assembler as the DIAGNOSE instruction.

**Note:** There is no mnemonic for DIAGNOSE.

Rx, Ry are general purpose registers that contain operand storage addresses or return codes passed to the DIAGNOSE interface. If the registers contain addresses, those addresses must be real to the virtual machine issuing the DIAGNOSE.

CODE is a two-byte hexadecimal value that CP uses to determine what DIAGNOSE function to perform. The codes defined for the general VM/SP user are described in this section. The code must be a multiple of four. Codes X'00' through X'FC' are reserved for IBM use, and codes X'100' through X'1FC' are reserved for users. The privilege class for each code is indicated.

Because DIAGNOSE operates differently in a virtual machine than it does in a real machine, a program should determine that it is operating in a virtual machine before issuing a diagnose instruction, and prevent execution of a DIAGNOSE when in a real machine. The Store Processor ID (STIDP) instruction provides a program with information about the processor in which it is executing, including the processor version number. If STIDP is issued from a virtual machine, the version number will be X'FF' in the first byte of the CPUID field.

A virtual machine issuing a diagnose instruction should run with interrupts disabled. This prevents loss of status information pertaining to the diagnose operation such as condition codes and sense data.

**Note:** A DIAGNOSE instruction with invalid parameters may in some cases result in a specification exception or protection exception.

## DIAGNOSE Code X'00' -- Store Extended-Identification Code

Privilege class G

Execution of DIAGNOSE code X'00' allows a virtual machine to examine the VM/SP extended-identification code. For example, an OS/VS1 virtual machine issues a DIAGNOSE code X'00' instruction to determine if the version of VM/SP

under which it is executing supports the VM/VS Handshaking feature. If the extended-identification code is returned to VS1, VM/SP supports handshaking; otherwise, it does not.

***Entry Values***: The register specified as Rx contains the doubleword aligned virtual storage address where the VM/SP extended-identification code is to be stored. The Ry register contains the number of bytes to be stored entered as an unsigned binary number.

***Exit Values***: If the VM/SP system currently executing does not support the DIAGNOSE code X'00' instruction, no data is returned to the virtual machine. If it does support the DIAGNOSE code X'00' instruction, the following data is returned to the virtual machine (at the location specified by Rx):

| Field | Description | Characteristics |
|---|---|---|
| System Name | "VM/SP" | 8 bytes, EBCDIC |
| Version Number | The first byte is the version number, the second byte is the level, and the third byte is the PLC (Program Level Change) number. | 3 bytes, hexadecimal |
| Version Code | VM/SP executes the STIDP (Store Processor ID) instruction to determine the version code. | 1 byte, hexadecimal |
| MCEL | VM/SP executes the STIDP instruction to determine the maximum length of the MCEL (Machine Check Extended Logout) area. | 2 bytes, hexadecimal |
| Processor Address | VM/SP executes the STAP (Store Processor Address) instruction to determine the processor address. | 2 bytes, hexadecimal |
| Userid | The userid of the virtual machine issuing the DIAGNOSE. | 8 bytes, EBCDIC |
| Program Product Bit Map | Identifies the program products that are installed. Valid values and the program products each identifies are: | 8 bytes, hexadecimal |

```
Value                    Program Product
X'8000000000000000'      Basic System Extensions 2
X'C000000000000000'      System Extensions, Release 2
X'E000000000000000'      VM/System Product, Release 1
X'F000000000000000'      VM/System Product, Release 2
X'F800000000000000'      VM/System Product, Release 3
```

| | | |
|---|---|---|
| Time Zone Value | Represents the time zone differential in seconds from Greenwich Mean Time. | 4 bytes, hexadecimal |
| UNUSED | | 4 bytes, zeroes |

**Note:** The Time Zone Value is a signed hexadecimal fullword value in seconds. Negative values represent differentials west of Greenwich Mean Time and positive values represent differentials east of Greenwich Mean Time. If VM/SP is executing in a virtual machine, another 40 bytes, or less, of extended identification data is appended to the first 40 bytes described above. Up to five nested levels of VM/SP virtual machines are supported by this diagnose instruction resulting in a maximum of 200 bytes of data that can be returned to the virtual machine that initially issued the DIAGNOSE instruction.

Upon return, Ry contains its original value less the number of bytes that were stored.

***Completion and Condition Codes***: No completion code is returned, and the condition code remains unchanged.

## DIAGNOSE Code X'04' -- Examine Real Storage

Privilege class C or E

***Entry Values***: Execution of a DIAGNOSE Code X'04' allows a user to examine real storage. The register specified as Rx contains the virtual address of a list of CP (real) addresses to be examined. The Ry register contains the count of entries in the list. Ry+1 contains the virtual address of the result field. The result field contains the values retrieved from the specified real locations.

***Exit Values***: For each address in the list of CP addresses, VM/SP provides a fullword of data obtained from the specified address in real storage. VM/SP stores this data into the result field identified by the Ry+1 register.

There is a one-to-one correspondence between entries in the list of addresses and entries in the result field. For example, data obtained from the address in the first entry of the address list is stored in the entry of the result field, data obtained from the second entry of the address list is stored in the second entry of the result field, and so forth.

***Usage Notes***: The request and result tables must be in the same page of virtual storage, and that page must be resident in real storage, at the time the DIAGNOSE is executed. This is guaranteed if the instruction itself is also in the same page.

In the attached processor or multiprocessor environment, each processor has a prefix register to relocate addresses between 0 and 4095 to another page frame in main storage. The prefix register enables each processor to use a different page frame in order to avoid conflict with the other processor for such activity as interrupt code recording. Thus, the range 0 through 4095 refers to different areas of storage, depending upon which processor generates the address.

In attached processor mode, all references to main storage from either processor are handled as if they were made on the main processor. In multiprocessor mode,

references to main storage from either processor are handled as if they were made on the IPL processor. Existing user programs remain valid for performance data; they receive the statistics for the main (or IPL) processor.

References to the PSA of the attached processor (or non-IPL processor, in multiprocessor mode) may be made as follows: first, retrieve the value of PREFIXB, the value of the prefix register for the other processor (the attached processor in this case). Next, specify addresses that are the sum of the value of PREFIXB and the PSA displacement. References to 0 through 4095 are made by summing the value of PREFIXA and the PSA displacement to form the request address. Several system values that are processor independent are maintained in 0 through 4095, such as the restart PSW and the trace table vectors.

**Note:** If a reference is made to a real page frame that CP has determined to be disabled, results cannot be predicted. The CORETABLE entry corresponding to the real page address is checked and, if a disabled condition is found, the operation is terminated and a program check for a specification exception is presented to the virtual machine.

## DIAGNOSE Code X'08' -- Virtual Console Function

Privilege class G

DIAGNOSE Code X'08' enables a virtual machine running in supervisor state to issue CP commands. The virtual machine must specify the command, the command parameters, and whether CP is to return the command response to the user's terminal or to a buffer. In addition to returning the command response, CP sets a completion code in the Ry register and may set a condition code.

***Entry values***: When DIAGNOSE Code X'08' is issued, the Rx and Ry registers must be set up as follows:

Rx -- Rx must point to the character string in virtual storage that contains the CP commands and parameters. If the character string contains multiple commands, each command and its associated parameters must be separated from adjacent commands by the value X'15'.

Ry -- The high-order byte contains flag bits; the other three bytes specify, in bytes, the length of the CP commands and parameters. The maximum allowable length is 240 characters.

Set the flag bits as follows. If CP is to reject a password entered on the same line as a LINK command, set the high-order bit to a value of one (X'80'). CP rejects passwords only if the installation specified password suppression during system generation. If CP is to return the command response in a buffer, set the second flag bit to a value of one (X'40').

***Exit values***: If the Ry register contains the value X'00000000', the DIAGNOSE Code acts as a no-operation (NOP) instruction. As a consequence, the issuing virtual machine is placed into a CP-READ state.

If the command response is to be returned in a buffer, Rx and Ry cannot be consecutive registers nor can either be register 15. In addition, the Rx+1 and Ry+1 registers must be setup as follows:

Rx+1 -- Rx+1 must point to the buffer in virtual storage where CP is to return the command response.

Ry+1 -- Ry+1 must specify, in bytes, the length of the buffer. This value must not exceed 8192.

***Condition Codes***: If the command response is to be returned in a buffer, CP sets a condition code and returns information as follows:

| | |
|---|---|
| condition code 0 | The request was successful. The Rx+1 register points to the buffer that contains the command response. The Ry+1 register specifies the length of the response. |
| condition code 1 | The request was unsuccessful. The response does not fit into the buffer. The Ry+1 register contains a value that specifies how many bytes of the response would not fit into the buffer. |

***Completion Codes***: When CP returns to a program executing a DIAGNOSE Code X'08' instruction, the length value that was supplied in register Ry is replaced by the CP completion code value. This value is zero if the CP console function was successfully executed. If an error occurred, the completion code is the numeric value expressed in the message describing the error. For example, if error message DMKCFM045E is issued, CP sets a completion code of 45.

If the user has not specified a command response buffer, error messages and informational messages are generated according to the current values established by SET EMSG, SET IMSG, and SET MSG commands.

If a command response buffer is used, error and informational messages are always put into the buffer instead of being written to the console. Each line of the response is followed by a new line character (X'15'). If the buffer is not long enough to contain all of the response lines, only as many complete lines as can fit into the buffer are supplied, so the last character written into the response buffer by CP is always a new line character. Any unused portion of the response buffer is not changed. The setting of IMSG is ignored (it is considered always to be ON) and the setting of EMSG determines only whether the error message code is retained. (SET EMSG OFF is treated the same as SET EMSG ON; SET EMSG TEXT suppresses error message codes.) Messages controlled by SET MSG (such as "PUN file nnnn to ...") are not put into the command response buffer unless SET MSG ON is in effect.

The completion code values returned by CP are not affected by the values of EMSG and IMSG, or by the use of a command response buffer.

If CP is executing multiple commands and encounters an invalid command, processing stops and CP ignores the remaining commands.

Following are two examples showing how to specify DIAGNOSE Code X'08'. The first example shows how a program issues the QUERY FILES command. In this example the response is returned to the user's terminal. Note that in virtual storage environment, a load real address (LRA) instruction must be used to load the Rx register.

```
          LA*     6,CMMD
          LA      10,CMMDL
          DC      X'83',X'6A',XL2'0008'
            .
            .
            .
CMMD      DC      C'QUERY FILES'
CMMDL     EQU     *-CMMD
            .
            .
```

The second example shows how to specify a string of commands when multiple commands are to be issued.

```
          LA*     6,CMMD
          LA      10,CMMDL
          DC      X'83',X'6A',XL2'0008'
            .
            .
            .
CMMD      DC      C'QUERY FILES'
          DC      X'15'
          DC      C'PURGE PRINTER'
CMMDL     EQU     *-CMMD
```

**Notes:**

1. If you are in EC mode you must code a LRA instruction instead of a LA instruction if you are running a virtual storage system (for example, MVS) in a virtual machine and want to specify the address of the CMMD parameter.

2. The logical line editing characters (described in the *VM/SP Terminal Reference* and under the TERMINAL command in the *CP Command Reference for General Users*) are only recognized by CP when entered from a terminal, not when passed to CP via DIAGNOSE X'08'. Therefore a command such as #CP is not recognized by CP when issued via DIAGNOSE X'08' and results in error message "DMKCFC001E ?CP: COMMAND".

## DIAGNOSE Code X'0C' -- Pseudo Timer

Privilege class G

Execution of DIAGNOSE Code X'0C' causes CP to store four doublewords of time information in the user's virtual storage.

***Entry Values***: The register specified as Rx contains the address of the 32 byte area where the time information is to be stored. The address must be on a doubleword boundary. The information returned is in the format shown in Figure 28 on page 207.

The first eight bytes contain the Month/Day-of-Month/Year. The next eight bytes contain the time of day in Hours:Minutes:Seconds. The last 16 bytes contain the virtual and total processor time used by the virtual machine that issued the DIAGNOSE. The last 16 bytes are expressed as a doubleword, unsigned integer. The time is expressed in microseconds.

***Completion and Condition Codes***: No completion code is returned, and the condition code remains unchanged.

## DIAGNOSE Code X'10' -- Release Pages

Privilege class G

Pages of virtual storage can be released by issuing a DIAGNOSE Code X'10'. When a page is released, it is considered all zero.

***Entry values***: The register specified by Rx contains the address of the first page to be released, and the Ry register contains the address of the last page to be released. Both addresses must be on page boundaries. A page boundary is a storage address whose low order three digits, expressed in hexadecimal, are zero.

***Completion and Condition Codes***: No completion code is returned, and the condition code remains unchanged.

**Note:** Do not use DIAGNOSE Code X'10' to release noncontiguous storage; use DIAGNOSE Code X'64' for this purpose.

## DIAGNOSE Code X'14' -- Input Spool File Manipulation

Privilege class G

Execution of DIAGNOSE Code X'14' causes DMKDRDER to perform input spool file manipulation.

***Entry Values***: Depending upon the value of the function specified as Rx contains a buffer address, a copy count, or a spool file identifier. The Ry register, which must be an even register, contains either the virtual address of a spool input card reader or, if Ry+1 contains X'0FFF', a spool file ID number. Ry+1 contains a hexadecimal code indicating the file manipulation to be performed, and a flag with the optional size of the spool file block. The codes are:

| Code | Function |
|---|---|
| 0000 | Read next spool buffer (data record) |
| 0004 | Read next print spool file block (SFBLOK) |
| 0008 | Read next punch spool file block (SFBLOK) |
| 000C | Select a file for processing |
| 0010 | Repeat active file *nn* times |
| 0014 | Restart active file at beginning |
| 0018 | Backspace one record |
| 001C | Read next monitor spool file block |
| 0020 | Read next monitor spool record |
| 0024 | Read last spool buffer (active file) |
| 0FFE | Select next file not previously selected |
| 0FFF | Retrieve subsequent file descriptor |

**Notes:**

1. Subcodes X'001C' and X'0020' are the only subcodes of DIAGNOSE X'14' that can be used for monitor files.

2. For subcodes X'0000', X'0004', X'0008', X'000C', X'001C', and X'0020', held files are skipped.

***Condition Codes***: On return Ry+1 may contain error codes that further define a returned condition code of 3.

| Condition Code | Ry+1 | Error |
|---|---|---|
| 0 | | Data transfer successful |
| 1 | | End of file or if subcode X'0018' and file is at first record |
| 2 | | File not found |
| 3 | 4 | Device address invalid |
| 3 | 8 | Device type invalid |
| 3 | 12 | Device busy, reader not ready, or device is a real device |
| 3 | 16 | Fatal paging I/O error |
| 3 | 20 | Page already locked for I/O |
| 3 | 24 | File in use by system; probable paging or spooling error. |

### *Subcode X'0000'*

Rx = start address of fullpage virtual buffer
Ry = virtual spool reader address
Ry+1 = function subcode

The specified device is checked for a file activated via DIAGNOSE. If one is found, the next fullpage buffer is made available to the virtual machine via a call to DMKRPAGT. If a file is not found, the chain of reader files is searched for a file for the calling user and connected to the virtual device for further reading. If no file is found, virtual condition code 2 is set. When the end of an active file is reached, the device status settings are tested for "spool continuous." If not set, virtual condition code 1 is set, indicating end of file. If the device is set for continuous input, the active file is examined to determine whether or not it is a multiple-copy file. If it is, reading is restarted at the beginning of the file. If it is not, the file is closed via DMKVSUCR and the reader chain is searched for another input file. If no other file is found, virtual condition code 1 is set. A specific DIAGNOSE X'14' Subcode X'0000' must be issued to get the first spooled page again.

**Notes:**

1. Subcode X'0000' returns a 3 condition code if an active monitor file or CP dump file is found.

2. Issuing DIAGNOSE X'14' subcode X'0000' against a locked page causes the page to become unlocked.

### *Subcode X'0004'*

Rx = virtual address of an SFBLOK buffer
Ry = virtual spool reader address
Ry+1 = flag, optional size of SFBLOK in doublewords, and function subcode.

If the specified device is in use via DIAGNOSE, the VSPLCTL block is checked to see whether or not this is a repeated call for printer SFBLOKs. If it is, then the chain search continues from the point where the last SFBLOK was given to the virtual machine. In this case, cc = 1 is set when there are no more print files. If this is

the first call for an SFBLOK, or if there have been intervening calls for file reading, the spool input chain is searched from the beginning, and cc=2 is set if no files are found.

If the high-order byte of the subcode register (Ry+1) is zero, then only 13 doublewords of the SFBLOK are returned and the rest could be truncated. However, if bit zero of the register is on, then bits 2 to 7 specify the amount of data to be returned (in doublewords). If the actual SFBLOK is shorter, the extra space is filled with zeros.

**Note:** The virtual buffer specified via Rx must not cross a page boundary or a specification exception results.

### *Subcode X'0008'*

Rx = virtual address of an SFBLOK buffer
Ry = virtual spool reader address
Ry+1 = flag, optional size of SFBLOK in doublewords, and function subcode.

Processing for subcode X'0008' is the same as for subcode X'0004', except that only punch files are processed.

**Note:** For both subcode X'0004' and subcode X'0008', the format definition for a VM/SP SFBLOK can be found in the system macro library.

### *Subcode X'000C'*

Rx = file identifier of requested file
Ry = virtual spool reader address
Ry+1 = function subcode

The spool input chain is searched for the file specified. If it is not found, cc=2 is set. If it is found, the file is moved to the head of the chain so that it is the next file processed by any of the other functions.

### *Subcode X'0010'*

Rx = new copy count for the active file
Ry = virtual spool reader address
Ry+1 = function subcode

The specified device is checked for an active file. If no file is active, cc=2 is set. Otherwise, the copy COUNT for the file is set to the specified value, with a maximum of 255. If the specified count is not positive, a specification exception is generated.

### *Subcode X'0014'*

Rx = start address of virtual fullpage buffer
Ry = virtual spool reader address
Ry+1 = function subcode

The specified device is checked for an active file. If no active file is found, cc=2 is set. Otherwise, the VSPLCTL pointers are reset to the beginning of the file.

### *Subcode X'0018'*

Rx = start address of virtual fullpage buffer
Ry = virtual spool reader address
Ry+1 = function subcode

The specified device is checked for an active file. If no active file is found, cc=2 is set. Otherwise, the file is backspaced one record and the record is given to the user as in subcode X'0000'. If the file is already positioned at the first record, the first record is given to the user.

### *Subcode X'001C'*

Rx = virtual address of an SFBLOK buffer
Ry = virtual spool reader address
Ry+1 = flag, optional size of SFBLOK in doublewords, and function subcode.

Processing is the same as Subcode X'0008', except that only monitor spool files, as identified by the SFBMON flag is SFBFLAG2, can be handled.

### *Subcode X'0020'*

Rx = start address of virtual fullpage buffer
Ry = virtual spool reader address
Ry+1 = function subcode

Processing is the same as Subcode X'0000', except that only monitor spool files, as identified by the SFBMON flag in SFBFLAG2, can be handled.

### *Subcode X'0024'*

Rx = start address of virtual fullpage buffer
Ry = virtual spool reader address
Ry+1 = function subcode

The specified device is checked for an already active file. If there is one, the last fullpage buffer is made available to the virtual machine via a call to DMKRPAGT. If there is no active file, CC=2 is set.

### *Subcode X'0FFE'*

Rx = virtual address of a 252 byte buffer
Ry = code to further determine function
Ry+1 = flag, optional size of SFBLOK in doublewords, and function subcode.

If Ry code = 0, the next reader spool file that was not previously seen is selected and returns data[4] to the user's buffer.

If Ry code is not zero, the bit in the SFBLOK is be reset to indicate that the spool file was previously selected and data[4] from the first spool file is returned to the user. CC=1 is returned if no file is found.

[4] The data for the X'0FFE' and X'0FFF' subcodes of the DIAGNOSE X'14' are SFBLOK, 40 bytes of the 3800 data from the first SPLINK (if requested), the first CCW, the following TIC, and up to 136 bytes of associated data.

Subcode X'0FFE' waits for a file being used by a system function. If, however, the file is not available within the 250 millisecond time limit, a condition code of 3, RC of 24 is returned. This condition indicates system problems due to performance or errors in the spooling area.

### *Subcode X'0FFF'*

Rx = virtual address of a 252 byte buffer
Ry = spool file ID number
Ry+1 = flag, optional size of SFBLOK in doublewords, and function subcode.

If Ry is nonzero, the spool input chain is searched for a file with a matching ID number: If none is found or if one is found that is owned by a different virtual machine, cc=2 is set. The chain search is continued from the file that was found, or from the anchor if Ry is zero, for the next file owned by the caller, independent of file type, class, etc. If none is found, cc=1 is set. If a file is found but it has the INUSE flag on, cc = 3 (rc = 12) is returned. Otherwise, the data[4] is returned to the user's buffer.

As with subcode X'0FFE', subcode X'0FFF' also waits for a file being used by a system function. If, however, the file is not available within the 250 millisecond time limit, a condition code of 3, RC of 24 is returned. This condition indicates system problems due to performance or errors in the spooling area.

**Note:** Data chaining may occur when 3800 load CCW's are present in a spool file. If the data following a 3800 load CCW is more than 4080 bytes long, that data cannot be contained in one DASD spool file buffer. Instead, the CCW is data-chained to succeeding DASD buffers until all the data has been entered into the spool file. If the file contains 3800 load CCW's, either the SFBLDBEG or the SFBLDMID flags are set in the SFBLOK.

The amount of SFBLOK data returned is calculated as described under subcode X'0004'. In addition, if bit zero of the subcode register (Ry+1) is on, 40 bytes of 3800 data is returned immediately following the SFBLOK and preceding the TAG data. The data returned is described in the SPLINK DSECT starting at label SPCHAR.

## DIAGNOSE Code X'18' -- Standard DASD I/O

Privilege class G

Input/output operations to a direct access device, of the type used by CMS, can be performed from a virtual machine using DIAGNOSE Code X'18'. No I/O interrupts are returned by CP to the virtual machine; the DIAGNOSE instruction is completed only when the READ or WRITE commands associated with the DIAGNOSE are completed[5].

***Entry Values***: The Rx register contains the virtual device address of the direct access device. The Ry register contains the address of a chain of CCWs. The user must load Register 15 with the number of READs or WRITEs in the CCW chain. The CCW chain must be in a standard format that CP expects when DIAGNOSE Code X'18' is used, as shown below.

---

5 For non-standard channel programs (more than one consecutive READ or WRITE CCWs chained together), no extended CCW is transformed if this is directed to a 3380 with the Speed Matching Buffer.

*Usage*: Diagnose X'18' checks that the byte count from the user's read or write CCW does not exceed 4096 bytes. If the byte count exceeds 4096 bytes, CP flags it as an error. If the byte count is less than or equal to 4096 bytes, Diagnose X'18' makes an additional check for valid CMS standard blocksizes. The standard CMS blocksizes are 512, 800, 1K, 2K, or 4K bytes. The latter check is necessary and only pertinent in the event that the user's channel program is directed to a device that is capable of executing extended count-key-data channel commands (for example, a 3380 attached to a 3880 Control Unit equipped with the Speed Matching Buffer Feature).

CP converts user's channel programs to the extended count-key-data (CKD) format when they:

- Are directed to a 3380 attached to a 3880 Control Unit equipped with the Speed Matching Buffer (Feature #6550)
- Or are directed to a 3375 attached to a 3880 Control Unit equipped with the Speed Matching Buffer (Feature #6560)
- And contain READ or WRITE CCW's with valid CMS blocksizes
- And contain no READs chained to READs or WRITEs which are, themselves, chained to WRITEs.

An example of a channel program converted to an extended count-key-data channel program is shown below.

DIAGNOSE X'18' must not be used to read or write record-overflow-formatted data.

A typical CCW string to read or write two 800-byte records is as follows:

```
   SEEK,A,CC,6
   SET SECTOR (not used for 2314/2319)
   SRCH,A+2,CC,5
   TIC,*-8,0,0
   RD or WRT,DATA,CC+SILI,800
   SEEK HEAD,B,CC,6 (omitted if HEAD number unchanged)
   SET SECTOR
   SRCH,B+2,CC,5
   TIC,*-8,0,0
   RD or WRT,DATA+800,SILI,800

A  SEEK and SRCH arguments for first RD/WRT
B  SEEK and SRCH arguments for second RD/WRT
```

If you are reading from or writing to either a 3380 or 3375 attached to a 3880 Control Unit equipped with the respective Speed Matching Buffer, the above sample channel program would be converted to the following extended count-key-data CCWs:

```
DEFINE EXTENT,C,CC,16
LOCATE RECORD,D,CC,16
RD OR WRT,DATA,CC+SILI,800
LOCATE RECORD,E,CC,16
RD OR WRT,DATA+800,SILI,800

C = DEFINE EXTENT argument
D = LOCATE RECORD argument for first RD/WRT
E = LOCATE RECORD argument for second RD/WRT
```

**Note:** The second LOCATE RECORD CCW shown in this example is not generated in all cases. That is, LOCATE RECORD CCWs, after the first one, are generated only when one of the following is encountered:

- A READ is followed by a WRITE, or vice versa, with the normal SEEK, SET SECTOR, SRCH in between them.
- The length of a READ or WRITE is not the same as the length of the preceding READ or WRITE.
- The READ or WRITE that follows a previous READ or WRITE is not for the next sequential record on the track.

***Condition and Completion Codes***: The codes returned are as follows:

cc=0 I/O complete with no errors

cc=1 Error condition. Register 15 contains one of the following:

R15=1 Device not attached

R15=2 Device not 2319, 2314, 3330, 3340, 3350, 3375, or 3380

R15=3 Attempt to write on a read-only disk

R15=4 Cylinder number not in range of user's disk

R15=5 Virtual device is busy or has an interrupt pending

R15=6 Device halted; I/O may or may not have completed.

cc=2 Error condition. Register 15 contains one of the following:

R15=5 Pointer to CCW string not doubleword-aligned.

R15=6 SEEK/SEARCH arguments not within range of user's storage.

R15=7 CCW is not a SEEK, SEEK HEAD, SET SECTOR, SEARCH ID, TIC*-8, READ, or WRITE or an invalid CCW string was submitted.

R15=8 READ/WRITE byte count=0

R15=9 READ/WRITE byte count greater than 4096

R15=10 READ/WRITE buffer not within user's storage

R15=11 The value in R15, at entry, was not a positive number from 1 through 15, or was not large enough for the given CCW string.

R15=12 Cylinder number on seek head was not the same number as on the first seek.

cc=3 Uncorrectable I/O error:

R15=13 CSW (8 bytes) returned to user Sense bytes are available if the user issues a SENSE command.

**Note:** This code does not support fixed-block DASD devices. If a program issues a DIAGNOSE Code X'18' to a fixed-block DASD device, CP sets cc=1 and places a return code of 2 in register 15.

## DIAGNOSE Code X'1C' -- Clear Error Recording Cylinders

Privilege class F

Execution of DIAGNOSE Code X'1C' allows a user to clear the error recording data on disk. The DMKIOEFM routine performs the clear operation.

***Entry Values***: The register specified as Rx contains a one-byte code value in the low-order byte as follows:

| *Code* | *Function* |
|---|---|
| X'01' | Clear and reformat all error recording, leaving any frame records intact |
| X'02' | Clear and reformat all error recording cylinders, erasing both frame records and error records |

## DIAGNOSE Code X'20' -- General I/O

Privilege class G

With DIAGNOSE Code X'20', a virtual machine user can specify any valid CCW chain to be performed on a tape, disk (including FBA) or unit record device. (An exception: DIAGNOSE must not be used to read or write record-overflow-formatted data on DASD devices.) No I/O interrupts are reflected to the virtual machine; the DIAGNOSE instruction is completed only when all I/O commands in the specified CCW chain are finished.

***Entry Values***: The register specified as Rx contains the virtual device address. The Ry register contains the address of the CCW chain, and CP uses the high-order byte of the register as a storage key for accessing the user's virtual storage.

The CCWs are processed via DMKCCWTR through DMKGIOEX, providing full virtual I/O in a synchronous fashion (self-modifying CCWs are not permitted, however) to any virtual machine specified. Control returns to the virtual machine only after completion of the operation or detection of a fatal error condition. EREP support is provided for tape and DASD devices only; all other devices present an error condition in the PSW to the virtual user. Condition codes and error codes are returned to the virtual system.

***Completion and Condition Codes***: The condition codes and error codes returned are as follows:

cc=0 I/O completed with no errors

cc=1 Error condition. Register 15 contains the following:

R15=1 Device is either not attached or the virtual channel is dedicated.

R15=5 Virtual device is busy or has an interrupt pending.

R15=6 Device halted; I/O may or may not have completed.

cc=2 Exception conditions. Register 15 contains one of the following:

R15=2 Unit exception bit in device status byte=1

R15=3 Wrong length record detected.

cc=3 Error Condition:

R15=13 A permanent I/O error occurred or an unsupported device was specified. The user's Ry register contains four sense bytes. Sense bytes 2 and 3 are in the two leftmost positions in the Ry register; sense byte 0 and 1 are in the two rightmost positions in the Ry register.

***Ry Register***

| Sense Byte 2 | Sense Byte 3 | Sense Byte 0 | Sense Byte 1 |
|---|---|---|---|

## DIAGNOSE Code X'24' -- Device Type and Features

Privilege class G

DIAGNOSE Code X'24' requests CP to provide a virtual machine with identifying information and status information about a specified virtual device. The virtual machine must specify the virtual device for which information is requested. CP returns information about the virtual device and associated real device in the Rx, Ry, and Ry+1 registers. CP also provides a condition code identifying the specific device information returned to the virtual machine.

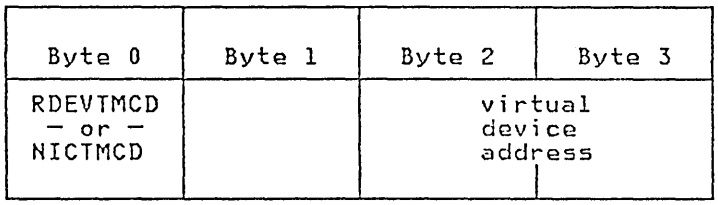
***Entry Values***: When a virtual machine issues DIAGNOSE Code X'24', the Rx register must contain the virtual device address for which information is requested or the value negative 1 (-1). Specify -1 when the device is a virtual console whose address is unknown to the virtual machine.

***Exit Values***: When CP returns control to the virtual machine, the Ry, Ry+1, and Rx registers contain device information. The Ry register contains information about the virtual device and the Ry+1 register information about the real device. If -1 was specified and CP located the virtual console, the Rx register contains information about the virtual console.

CP obtains device information from three control blocks: virtual device information from the virtual device block (VDEVBLOK), and real device information from the real device block (RDEVBLOK) and from NICBLOK. The following diagrams identify specific information returned by CP and show how to locate this information in the Rx, Ry, and Ry+1 registers. The symbolic names used in these diagrams are the symbolic names used with VDEVBLOK, RDEVBLOK, and NICBLOK in *VM/SP Data Areas and Control Block Logic, Volume 1.*

**Note:** For a DIAGNOSE X'24' to an SNA device though VCNA, the model (RDEVMDL) information is correct, however, the RDEVTYPE may not be reliable.

***Rx Register***




| *Symbolic Name* | *Meaning* |
|---|---|
| RDEVTMCD<br>- or -<br>NICTMCD | Terminal code bits defining the type of console and the translate table the console is using. RDEVTMCD is for a local virtual console; NICTMCD for a remote 3270 virtual console. |

***Ry Register***

| Byte 0 | Byte 1 | Byte 2 | Byte 3 |
|---|---|---|---|
| VDEVTYPC | VDEVTYPE | VDEVSTAT | VDEVFLAG |

| *Symbolic Name* | *Meaning* |
|---|---|
| VDEVTYPC | Virtual device type class |
| VDEVTYPE | Virtual device type |
| VDEVSTAT | Virtual device status |
| VDEVFLAG | Virtual device flags |

*Ry+1 Register*

| Byte 0 | Byte 1 | Byte 2 | Byte 3 |
|---|---|---|---|
| RDEVTYPC | RDEVTYPE<br>- or -<br>NICDTYPE | RDEVMDL<br>- or -<br>NICMDL | RDEVFTR<br>- or -<br>RDEVLLEN<br>- or -<br>NICLLEN |

| *Symbolic Name* | *Meaning* |
|---|---|
| RDEVTYPC | Real device type class |
| RDEVTYPE | Real device type |
| RDEVMDL | Real device model number. To determine if the speed matching buffer for the 3380 or 3375 is present, check if bits 0 and 1 are set on. |
| RDEVFTR | Real device feature code for a device other than a virtual console |
| RDEVLLEN | Current device line length for a local virtual console |
| NICDTYPE | Real device type for a remote 3270 virtual console |
| NICMDL | Real device model number for a remote 3270 virtual console |
| NICLLEN | Current device line length for a remote virtual console |

***Condition Codes***: The following chart lists the condition codes CP can return for DIAGNOSE Code X'24', the meaning of each condition code, and the registers where data is returned.

| If the condition code equals | This register contains information | | | Comments |
|---|---|---|---|---|
| | Rx[1] | Ry | Ry+1[2] | |
| 0 | X | X | X | Normal completion |
| 1 | | | | Undefined |
| 2 | X | X | | The virtual device exists but is not associated with a real device |
| 3 | | | | Invalid device address or the virtual device does not exist |

[1]The Rx register contains information only when DIAGNOSE Code X'24' specifies a virtual console whose address is unknown.
[2]If Ry is register 15, CP returns only virtual device information; no information is returned in register Ry+1.

## DIAGNOSE Code X'28' -- Channel Program Modification

Privilege class G

DIAGNOSE Code X'28' allows a virtual machine to correctly execute some channel programs modified after the Start I/O (SIO) instruction is issued and before the input/output operation is completed. The channel command word (CCW) modifications allowed are:

- A Transfer in Channel (TIC) CCW modified to a No Operation (NOP) CCW
- A TIC CCW modified to point to a new list of CCWs
- A NOP modified to a TIC CCW.

When a virtual machine modifies a TIC CCW, it is modifying a virtual channel program. CP has already translated that channel program and is waiting to execute the real CCWs. The DIAGNOSE instruction, with Code X'28', must be issued to inform CP of the change in the virtual channel program, so that CP can make the corresponding change to the real CCW before it is executed. In addition, when a NOP CCW is modified to point to a new list of CCWs, CP translates the new CCWs.

To be sure that the DIAGNOSE instruction is recognized in time to update the real CCW chain, the virtual machine issuing the DIAGNOSE instruction should have a high favored execution value and a low dispatching priority value. The CP SET command should be issued:

```
SET FAVORED xx

SET PRIORITY nn
```

where xx has a high numeric value and nn has a low numeric value. The virtual machine issuing the DIAGNOSE Code X'28' must be in the supervisor mode at the time it issues the DIAGNOSE instruction.

***Entry Values***: When DIAGNOSE Code X'28' is issued, the Rx register contains the address of the TIC or NOP CCW that was modified by the virtual machine. The Ry register contains the device address in bits 16 through 31. Rx and Ry cannot be the same register. The addresses specified in the Rx register, the new address in the modified TIC CCW, and the new CCW list to which the modified TIC CCW points must all be addresses that appear real to the virtual machine: CP knows these addresses are virtual, but the virtual machine thinks they are real.

***Condition and Completion Codes***: The condition codes (cc) and completion codes are as follows:

cc=0 The real channel program was successfully modified; register 15 contains a zero.

cc=1 There was probably an error in issuing the DIAGNOSE instruction. Register 15 (R15) contains one of the following completion codes:

R15=1 The same register was specified for Rx and Ry.

R15=2 The device specified by the Ry register was not found.

R15=3 The address specified by the Rx register was not within the user's storage space.

R15=4 The address specified by the Rx register was not doubleword aligned.

R15=5 A CCW string corresponding to the device (Ry) and address (Rx) specified was not found.

R15=6 The CCW at the address specified by the Rx register is not a TIC nor a NOP, or the CCW in the channel program is not a TIC nor a NOP.

R15=7 The new address in the modified TIC CCW is not within the user's storage space.

R15=8 The new address in the modified TIC CCW is not doubleword aligned.

R15=11 The modified CCW may not be a NOOP with command chaining if it is the last CCW in the real chain program.

cc=2 The real channel program cannot be modified because a channel end or device end already occurred. Register 15 contains a 9. The virtual machine should restart the modified channel program.

## DIAGNOSE Code X'2C' -- Return DASD Start of LOGREC

Privilege class C, E, or F

Execution of DIAGNOSE Code X'2C' allows a user to find the location on the disk of the error recording area, the number of error recording cylinders, and the location of the first error record.

***Entry Values***: The register specified as Rx contains a one-byte code in the low-order byte, indicating the function to be performed:

X'01' Return the DASD location of the start of the error recording area, and the number of error recording cylinders.

X'02' Return the HDRSTART value (DASD location of first error record).

X'04' Return indication of whether there are frame records on the error recording cylinders.

***Exit Values***: On return to the issuer of DIAGNOSE Code X'2C':

If code '01' is specified: Register Rx contains the DASD location (in VM/SP control program internal format) of the start of the error recording area. Ry contains, in the low-order halfword, the number of error recording cylinders.

If code '02' is specified: Register Rx contains the DASD location of the first error record (in CCPD format). The value actually points to the last frame record written, or record 2 if no frame records present.

If code '04' is specified: Register Ry contains a X'02' in the low-order byte if frame records are present on the error recording cylinders; X'00' if no frame records present.

**Note:** Codes '02' and '04' may both be specified (code '06') on invoking DIAGNOSE. Both an Rx and Ry value must be specified.

## ıGNOSE Code X'30' -- Read One Page of LOGREC Data

Privilege class C, E, or F

Execution of DIAGNOSE Code X'30' allows a user to read one page of the system error recording area.

***Entry Values***: The register specified as Rx contains the DASD location (in VM/SP control program internal format) of the desired record. The Ry register contains the virtual address of a page-size buffer to receive the data. The DMKRPAGT routine supplies the page of data.

***Condition Codes***: The condition codes returned are:

| Condition Code | Meaning |
|---|---|
| 0 | Successful read, data available |
| 1 | End of area, no data |
| 2 | I/O error |
| 3 | Invalid location, outside recording area |

**Note:** Issuing DIAGNOSE X'30' against a locked page causes the page to become unlocked.

## DIAGNOSE Code X'34' -- Read System Dump Spool File

Privilege class C or E

A user can read the system spool file by issuing a DIAGNOSE Code X'34' instruction. However, this Diagnose Code cannot read spool files that contain VMDUMP records -- use DIAGNOSE Code X'14' for this purpose. If a program attempts to use DIAGNOSE Code X'34' to read VMDUMP records, CP returns a condition code of 2.

***Entry Values***: The register specified as Rx contains the virtual address of a page-size buffer to receive the data. The Ry register, which must not be register 15, contains the virtual address of the spool input card reader.

***Condition Codes***: Ry+1, on return, may contain error codes as follows:

| Condition Code | Ry+1 Error Code | Meaning |
|---|---|---|
| 0 | | Data transfer successful |
| 1 | | End of file |
| 2 | | File not found |
| 3 | 4 | Device address invalid |

| Condition Code | Ry+1 Error Code | Meaning |
|---|---|---|
| 3 | 8 | Device type invalid |
| 3 | 12 | Device busy |
| 3 | 16 | Fatal paging I/O error |

The DMKDRDMP routine searches the system chain of spool input files for the dump file belonging to the user issuing the DIAGNOSE instruction. The first (or next) record from the dump file is provided to the virtual machine via DMKRPAGT and the condition code is set to zero. The dump file is closed via VM/SP console function CLOSE.

Note: Issuing DIAGNOSE X'34' against a locked page causes the page to become unlocked.

## DIAGNOSE Code X'38' -- Read System Symbol Table

Privilege class C or E

Execution of DIAGNOSE Code X'38' causes the routine DMKDRDSY to read the system table into storage.

***Entry Values***: The register specified as Rx contains the address of the page buffer to contain the symbol table.

***Condition Codes***: When complete, the Ry register, which must not be register 15, contains a condition code. On return, Ry+1 may contain an error code.

| Condition Code | Ry+1 Error Code | Meaning |
|---|---|---|
| 0 | | Full page of data available to virtual machine |
| 1 | | No symbol table is available |
| 3 | | Page buffer is locked for an I/O operation |
| 3 | 16 | Fatal paging I/O error |

**Notes:**

1. The format of the symbol table entries is described in CP macro SYM.
2. Issuing DIAGNOSE X'34' against a locked page causes the page to become unlocked.

## DIAGNOSE Code X'3C' -- VM/SP Directory

Privilege class A, B, or C

Execution of DIAGNOSE Code X'3C' allows a user to dynamically update the VM/SP directory. The register specified as Rx contains the first 4 bytes of the volume identification. The first two bytes of Ry contain the last two bytes of the volume identification. The last two bytes of Ry contain the volume address. The routine DMKUDRDS dynamically updates the directory.

***Condition Codes***: The PSW condition code is set depending on the success of the operation or the meaning of the condition code. The condition codes are set as follows:

| Condition Code | Meaning |
|---|---|
| 0 | Operation is successful. |
| 2 | Volume not found, not mounted, or not a valid directory volume. |
| 3 | Fatal I/O error trying to read the directory |

## Diagnose Code X'40' -- Clean-Up after Virtual IPL by Device

Privilege class G

This code is valid only during virtual IPL. Clean-up restores the user's page and frees the real page if it is not in the V=R machine. If the real page is in the V=R machine, the real page is not freed. The PSW from location zero of the virtual machine is loaded and made the current PSW.

***Entry Values***: Register Rx must contain a zero. Register Ry must point to the virtual machine registers to be loaded.

***Usage***: If the user issues a DIAGNOSE code X'40' outside of its use in DMKVMI, a specification exception is returned.

## DIAGNOSE Code X'48' -- Issue SVC 76 from a Second Level VM/370 or VM/SP Virtual Machine

Privilege class G

A second level VM/370 or VM/SP operating system issues SVC 76 using this DIAGNOSE. SVC 76 handles I/O error recording for virtual operating systems. For instance, a virtual machine issues SVC 76 to record data about hardware errors that occur on devices dedicated to it.

***Entry Values***: R1 is the Rx register. The Ry register is not used in this DIAGNOSE. R1 must contain either of two values:

X'04' -indicates an SVC 76 request from a VM/370 or VM/SP virtual machine

X'08' -indicates that a VM/370 or VM/SP virtual machine issued DIAGNOSE X'48'.

***Usage***: CP checks first for the X'04' value. If it is present, CP sets VMSPMFLG in the virtual machine's VMBLOK to X'04' and processes the SVC 76 request on behalf of the virtual machine.

If R1 contains a X'08' value, CP sets VMSPMFLG in the virtual machine's VMBLOK to X'08'. It then reflects the SVC 76 back to the virtual machine. The virtual machine then handles its own error recording.

For more information on SVC 76 and I/O error recording procedures, refer to *VM/SP OLTSEP and Error Recording Guide.*

## DIAGNOSE Code X'4C' -- Generate Accounting Records for the Virtual User

Privilege class G

This code can be issued only by a user with the account option (ACCT) in his directory.

***Entry Values***: Rx contains the virtual address of either a 24-byte parameter list identifying the "charge to" user, or a variable length data area that is to be stored in the accounting record. The interpretation of the address is based on a hexadecimal code supplied in Ry. If the virtual address represents a parameter list, it must be doubleword aligned; if it represents a data area, the area must not cross a page boundary. If Rx is interpreted as pointing to a parameter list and the value in Rx is zeros, the accounting record is spooled with the identification of the user issuing the DIAGNOSE instruction.

Ry contains a hexadecimal code interpreted by DMKHVC as follows:

| Code | Rx points to: |
|---|---|
| 0000 | a parameter list containing only a userid. |
| 0004 | a parameter list containing a userid and account number. |
| 0008 | a parameter list containing a userid and distribution number. |
| 000C | a parameter list containing a userid, account number, and distribution number. |
| 0010 | a data area containing up to 70 bytes of user information to be transferred to the accounting card starting in column 9. |

**Notes:**

1. For code X'0010', the only valid accounting record identification code (ACNTCODE field of the ACNTBLOK) is "C0". For the other four codes listed above, the accounting record identification code can be "C1", "C02", etc. For more information on accounting record identification codes, see *VM/SP Data Areas and Control Block Logic, Volume 1*.

2. If Ry contains X'0010', Ry cannot be register 15.

Ry+1 contains the length of the data area pointed to by Rx. If Rx points to a parameter list (Ry not equal to X'0010'), Ry+1 is ignored.

DMKHVC checks the VMACCOUN flag in VMPSTAT to verify that the user has the account option and if not, returns control to the user with a condition code of one.

If Ry contains a code of X'0010', DMKHVC performs the following checks:

- If the address specified in Rx is negative or greater than the size of the user's virtual storage, an addressing exception is generated.
- If the combination of the address in Rx and the length in Ry+1 indicates that the data area crosses a page boundary, a specification exception is generated.
- If the value in Ry+1 is zero, negative, or greater than 70, a specification exception is generated.

When Ry contains a code of X'0010', and if both the virtual address and the length are valid, DMKFREE is called to obtain storage for an account buffer (ACNTBLOK) which is then initialized to blanks. The userid of the user issuing the DIAGNOSE instruction is placed in columns 1 through 8 and an accounting record identification code of "C0" is placed in columns 79 and 80. The user data pointed to by the address in Rx is moved to the accounting record starting at column 9 for a length equal to the value in Ry+1. A call to DMKACOQU collects the accounting records on the system accounting chain (DMKRSPAC) and puts them in spool format. DMKHVC then returns control to the user with a condition code of zero.

If Ry contains other than a X'0010' code, control is passed to DMKCPV to generate the record. DMKCPV passes control to DMKACO to complete the "charge to" information; either from the User Accounting Block (ACCTBLOK), if a pointer to it exists, or from the user's VMBLOK. DMKCPV passes control back to DMKHVC to release the storage for the ACCTBLOK, if one exists. DMKHVC then checks the parameter list address for the following conditions:

- If zero, control is returned to the user with a condition code of zero.
- If invalid, an addressing exception is generated.
- If not aligned on a doubleword boundary, a specification exception is generated.

For a parameter list address that is nonzero and valid, the userid in the parameter list is checked against the directory list and if not found, control is returned to the user with a condition code of two. If the function hexadecimal code is invalid, control is returned to the user with a condition code of three. If both userid and function hexadecimal code are valid, the User Accounting Block (ACCTBLOK) is built and the userid, account number, and distribution number are moved to the block from the parameter list or the User Machine Block belonging to the userid in the parameter list. Control is then passed to the user with a condition code of zero.

## DIAGNOSE Code X'50' -- Save the 370X Control Program Image

Privilege class A, B, or C

DIAGNOSE Code X'50' invokes the CP module DMKSNC to validate the parameter list and write the page-format image of the 370X control program to the appropriate system volume.

When a 370X control program load module is created, the CMS service program SAVENCP builds a communications controller list (CCPARM) of control information. It passes this information to CP via a DIAGNOSE Code X'50'.

***Entry Values***: The register specified as Rx contains the virtual address of the parameter list (CCPARM). The Ry register is ignored on entry.

***Exit Values***: Upon return, the Ry register contains the following error codes:

| Code | Meaning |
|---|---|
| 044 | 'ncpname' was not found in system name table. |
| 171 | System volume specified not currently available. |
| 178 | Insufficient space reserved for program and system control information. |
| 179 | System volume specified is not a CP-owned volume. |
| 435 | Paging error while writing saved system. |

## DIAGNOSE Code X'54' -- Control The Function of the PA2 Function Key

Privilege class G

DIAGNOSE Code X'54' controls the function of the PA2 function key. The PA2 function key can be used either to simulate an external interrupt to a virtual machine or to clear the output area of a display screen.

***Entry Values***: The function performed depends upon how Rx is specified when DIAGNOSE Code X'54' is issued. If Rx contains a nonzero value, the PA2 key simulates an external interrupt to the virtual machine. If Rx contains a value of zero, the PA2 key clears the output area of the display screen.

***Usage***: The external interrupt is simulated only when the display screen is in the VM READ, HOLD, or MORE status and the TERMINAL APL ON command has been issued.

## DIAGNOSE Code X'58' -- 3270 Virtual Console Interface

Privilege class G

DIAGNOSE Code X'58' enables a virtual machine to communicate with 3270 display stations. Using DIAGNOSE Code X'58', a virtual machine may:

- Display up to a full screen of data using only one write operation.
- Provide attribute characters along with data that is sent to the display station. An attribute character provides control information for the data, for example, a request to intensify the data when it is displayed.
- Place a 3270 display station under control of the virtual machine (full screen mode).

***Entry Values***: When a virtual machine issues DIAGNOSE Code X'58', the virtual machine must provide one or more channel command words (CCWs). These CCWs specify the 3270 operation to be performed, provide control information for the display station, and specify the address of data to be displayed during a write operation or the address of a buffer where data is to be stored during a read operation. If only one CCW is used, the Rx register must contain its address. If CCWs are chained, the Rx register must contain the address of the first CCW in the chain. The Ry register must contain the virtual address of the display station where the operation is to be performed. This value must be right-justified.

### *Displaying Data*

To display up to a full screen of data, code a CCW using the following assembler language instructions:

```
DS 0D
DC AL1(CCWCODE),AL3(DATADDR),AL1(FLAGS),AL1(CTL),AL2(COUNT)
```

***where:***

CCWCODE is the command code X'19'.

DATADDR is the virtual storage address of the first byte of data to be displayed.

FLAGS are standard CCW flags. The suppress-incorrect-length indicator, bit 34, must be set to a value of one. Set other bits as needed.

CTL is a control byte defined as follows:

- The high-order bit (0), if set on, enables the screen "MORE" status to be active before the displaying of data.
- Bits 2-7 identify the line on the display screen where the display is to start. A value of 0 (B'xx00 0000') corresponds to the first or top line, a value of 1 (B'xx00 0001') corresponds to the second line and so forth.
- If the control byte contains the value X'FF', CP erases the display station's screen. No new data is displayed.
- CCW's may be command chained to combine several operations in one DIAGNOSE. When CP builds the real CCW string, it will data chain as many CCW's as possible to reduce the number of real I/O operations. If the control byte contains a value of X'FE', CP will:
  - Not data chain this operation to any previous CCW in the real CCW string.
  - Erase the entire screen.
  - Rewrite the attribute bytes for the CP screen format.
  - Reset the cursor to the beginning of the input area.

COUNT specifies the number of bytes of data to be displayed. The maximum that can be specified for this command code is 2032 bytes. The maximum amount of data that can be displayed at one time depends upon the 3270 model of the display station:

- A model 2 can display up to 1760 bytes
- A model 3 can display up to 2400 bytes
- A model 4 can display up to 3280 bytes
- A model 5 can display up to 3300 bytes

To provide attribute characters for the data, place the attribute character in the data stream immediately following a 3270 start-field order. The start-field order, a one-byte value, notifies the 3270 display system that the next byte in the data stream is an attribute character. For a description of how the 3270 display system uses attribute characters, and to determine the values to specify for attribute characters and the start-field order, see the *IBM 3270 Information Display System Library User's Guide.*

**Note:** Through the use of the attribute character, it is possible to define a display field as selector-pen detectable. However, when the selector pen is used to select the field, CP does not return data from the field to the virtual machine. After processing DIAGNOSE code X'58', CP sets a condition code. If the operation was successful - that is, no I/O errors occurred - CP sets a condition code of zero. If an I/O error occurred, CP sets a condition code of one.

If an I/O error occurred, the application program can check the I/O status and the error type by:

- Issuing a TEST I/O (TIO) instruction
- Examining the returned condition code
- Examining the virtual CSW

The returned condition codes and CSW status are the standard condition codes and status defined in the *IBM System/370 Principles of Operation.*

You must also make sure that the interrupt for the virtual device is enabled by setting the appropriate bit and channel mask in the PSW. For example, if the virtual address of your console is 009, bit 0 in the channel mask must be set to one (that is, bit 0 must be on). This may be the case if you are loading programs in the transient area.

### *Full Screen Mode*

DIAGNOSE X'58' provides a means by which a virtual machine may share, with CP, control of a 3270 display station. Two CCW operations, X'29' and X'2A', in addition to performing the requested I/O, notify CP that the display station is operating under the control of the virtual machine.

CCW code X'29' performs a WRITE, ERASE/WRITE, ERASE/WRITE ALTERNATE, or WRITE STRUCTURED FIELD operation, depending on the value of the control field. For the WRITE, ERASE/WRITE, and ERASE/WRITE ALTERNATE, the virtual machine must provide appropriate control information beginning with the Write Control Character (WCC) and including 3270 orders following the WCC. Data may be written anywhere on the screen. The virtual machine must provide the address where the write is to begin; it uses a SET BUFFER ADDRESS (SBA) order to do this. Writing can also start at the current cursor address.

CCW code X'29' performs a WRITE STRUCTURED FIELD operation when the value of the control field is X'20'. The WRITE STRUCTURED FIELD instruction sends control information to a 3274 controller. The application program must provide the control information in the data stream in the format required by the instruction. (See the *3270 Component Description* for more information on WRITE STRUCTURED FIELD operation.)

CCW code X'2A' performs a READ BUFFER or a READ MODIFIED operation, depending on the value of the control field.

To specify the full screen mode CCW, use the following assembler language instructions:

```
DS 0D
DC AL1(CCWCODE),AL3(DATADDR),AL1(FLAGS),AL1(CONTROL),AL2(COUNT)
```

***where:***

CCWCODE is a CCW code (X'29' or X'2A')

DATADDR for a write operation, specifies the first byte of the data stream (WCC) to be written. For a read operation, specifies the address of the read buffer.

FLAGS is the standard CCW flag field.

CONTROL for a write operation, an ERASE/WRITE is performed by specifying a ccwcode of X'29', if the high-order bit (X'80') is on. If the two high-order bits (X'C0') are on, an ERASE/WRITE ALTERNATE is performed. If bit 2 (X'20') is on, a WRITE STRUCTURED FIELD is performed.

If the high-order bit (X'80') is on for a read operation, a READ MODIFIED is performed; otherwise, a READ BUFFER is performed. The addition of X'10' to the CONTROL field values for ERASE/WRITE or ERASE/WRITE ALTERNATE, making them X'90' or X'D0' respectively, causes the PA1 key interrupt to be reflected to the virtual machine. This replaces the normal PA1 key function of returning the virtual machine to CP mode. This allows a virtual machine to have full control of the keyboard. Normal PA1 key function is restored when full screen mode is reset.

COUNT for a write operation, specifies the number of bytes to be displayed plus the number of bytes of control information. For a read operation, specifies the number of display characters to be read plus the number of bytes of control information. The maximum number of bytes that can be specified is 65503. The maximum number of displayable positions for the supported devices is:

3277 and 3275 Model 2 - 1920 bytes
3278, 3276 and 3279 Model 2 - 1920 bytes
3278, 3276 and 3279 Model 3 - 2560 bytes
3278 and 3276 Model 4 - 3440 bytes
3278 Model 5 - 3564 bytes

**Full Screen Interactions**

The virtual machine console exists in either of two modes, CP mode and full screen mode. CP mode is the default screen mode and is indicated by the screen status field in the lower right-hand corner of the screen. When in CP mode, the screen format is controlled by CP, and the data that appears on the screen is provided by CP and the programs running in the virtual machine. Full screen mode is initiated

by the application program running in the virtual machine. When in full screen mode, the screen format and data are under complete control of the program running in the virtual machine.

If TERMINAL BREAKIN GUESTCTL is specified, the screen mode changes only when the break-in key is used. An audible alarm is sounded when CP messages are queued. Priority CP messages and DIAGNOSE X'08' output take over the full screen.

CP mode is terminated and full screen mode is initiated when the application program issues an ERASE/WRITE instruction. Full screen mode may be terminated by a CP mode type I/O to the screen any time the keyboard is in a locked state.

Interactions between CP and the application program in the virtual machine using full screen support are listed below. The application programmer must be familiar with the operation of the IBM 3270 display station. For detailed information on its operation, see the appropriate 3270 Information Display System Description and Programmer's Guide listed in the Preface. Also listed below are general programming considerations that must be followed to effectively use the DIAGNOSE X'58' instruction for full screen I/O.

1. A full screen ERASE/WRITE or ERASE/WRITE ALTERNATE operation establishes full screen mode.

2. The application program is responsible for all I/O status and error checking, just as if START I/O (SIO) were being used instead of DIAGNOSE. This is done by using the TEST I/O (TIO) instruction and examining the returned condition code, and by examining the virtual CSW. The returned condition codes and CSW status are the standard condition codes and status as defined in the *IBM System/370 Principles of Operation,* with one exception noted below in number 5.

3. When in full screen mode, all CP messages are queued. The entire queue of CP messages is processed after each of the following operations:

   a. A full screen READ operation (any READ operation that locks the keyboard).

   b. A full screen WRITE operation that does not place the keyboard in the active status.

   c. The expiration of a 60 second timer for CP priority messages.

4. If a priority CP message (such as a warning message from the system operator) is to be displayed while in full screen mode, an attention interruption is posted to the application program and a 60 second timer is set. This informs the application program that a READ operation should be initiated. If a READ is not issued before the 60 seconds have expired, CP erases the screen and displays all queued messages.

   The only exception is if the application program has issued a Full Screen Support WRITE STRUCTURED FIELD instruction. CP does not take over the screen if the user has issued a WRITE STRUCTURED FIELD.

5. When a mode switch has occurred and the screen is in CP mode, the application program is notified by an X'8E' in the CSW unit status byte following a

full screen I/O operation. An ERASE/WRITE or ERASE/WRITE ALTERNATE instruction should be issued to reestablish full screen mode and reformat the screen. If control of the PA1 key interrupt had been transferred to the virtual machine via the CONTROL option, it must be respecified to return PA1 key control back to the virtual machine. Otherwise, depression of the PA1 key places the display in CP mode.

An X'8E' in the CSW unit status byte following an ERASE/WRITE or ERASE/WRITE ALTERNATE instruction indicates that non-full screen data (CP mode) is waiting to be read. The application program should issue a non-full screen READ and then reissue the ERASE/WRITE instruction.

6. Other non-full screen virtual machine messages are displayed immediately when in full screen mode.

7. The application program must establish an environment to handle attention interruptions. This could be done using the CMS macros HNDINT and WAITD. There are two conditions when CP posts an attention interruption to the application program:

   a. When CP receives an attention interruption indicating that the virtual machine console operator has caused an interruption. (For example, depressed the ENTER or a PF key on the display keyboard).

   b. When a CP priority message is to be displayed. In either case the application program should respond by issuing a READ.

8. The application program must also establish an environment to handle I/O interruptions and must ensure that channel end and device end have been received before processing continues.

9. If the test request key is depressed from a local 3270 when in full screen mode, X'604040' is returned to the application program in the read buffer. The test request key is not supported for remote 3270 terminals.

10. If you press the PA1 key in full screen mode, CP posts an attention interrupt to your virtual machine. If the virtual machine does not respond with a READ and you press the PA1 key a second time, your virtual machine is put in CP mode and "CP READ" is displayed in the screen's status area. However, if you set bit X'10' of the *control* option on before the initial ERASE/WRITE or ERASE/WRITE ALTERNATE, and press the PA1 key, the interrupt is reflected to your virtual machine for handling. If you have not set bit X'10' of the CONTROL option on and you press the PA1 key, your virtual machine is put in CP mode and "CP READ" is displayed in the screen's status area.

11. The application programmer must be aware that long data streams may result in very high CP storage use and possible system degradation. In addition, long data streams sent over BSC lines may cause degradation of response time on other terminals on the same BSC line.

### Full Screen Interactions (3270 SIO)

Full screen console (3270 SIO) support enables a guest virtual machine and CP to share a locally attached display terminal controlled by CP. The virtual machine can use the display terminal as a graphics device in full screen mode; CP can use the same terminal as a line device. When the terminal is in full screen mode, the screen

format, data checking, and error checking are under the complete control of the application program running in the virtual machine. A guest virtual machine can use either the DIAGNOSE X'58' or the SIO instruction to initiate full screen mode, but not both.

Before the guest virtual machine can issue 3270 SIO commands, it must first issue the CP TERMINAL command with the CONMODE 3270 option to be able to issue 3270 SIO commands. In addition, the SCRNSAVE ON option of the CP TERMINAL command gives a virtual machine (that has also specified CONMODE 3270) the ability to save the full screen display when the screen enters CP mode. If SCRNSAVE ON is specified, the screen is automatically redisplayed when the console returns to full screen mode. If SCRNSAVE OFF has been specified by a virtual machine that has specified CONMODE 3270 and CP takes over a screen, CP presents a CLEAR attention interrupt to the virtual machine when CP is ready to give up control of the screen. It is the responsibility of the application program to issue an ERASE/WRITE to refresh the screen. If the virtual machine issues only a WRITE that does not cover the entire screen, information that CP displayed can remain on the screen. To use the CP TERMINAL SCRNSAVE OFF:

1. Always issue a WRITE after a READ.

2. CP can break into a CCW chain containing WRITEs (with the WCC byte making the keyboard locked) and take over the screen. Upon return to full screen mode, the next CCW in the chain is processed as if it is the first CCW. The guest system must provide a means to handle this situation.

3. Refresh the screen with an ERASE/WRITE when CP issues a CLEAR attention interrupt.

4. When ATTENTION from the console is received, the guest program must issue a READ.

   The TERMINAL BREAKIN GUESTCTL option allows a guest operator to control break-ins (when CP takes over the full screen). Each time a CP request is received, it is put on a defer queue and an audible alarm sounds. The guest operator can switch to CP mode by hitting the break-in key.

   The TERMINAL BRKKEY option allows the user to specify a PF key as the break-in key in full screen mode. The default break-in key is PA1. PA1 attentions are sent to the virtual machine when PA1 is not defined as the BRKKEY. Some applications may interpret this PA1 attention as a user request to enter the CP environment.

**Notes:**

1. DIAGNOSE X'58' is a 3215 command and causes command rejects if executed with CONMODE 3270.

2. DIAGNOSE X'58' can be used with BREAKIN and BRKKEY.

3. CONMODE must be 3215 to run CMS. If CMS sets CONMODE to 3270 while CMS is running, results are unpredictable.

4. SCRNSAVE ON must be specified if running a guest SCP such as MVS with CONMODE 3270. Otherwise results are unpredictable.

5. CONMODE 3270 is not supported for disconnected users.

## DIAGNOSE Code X'5C' -- Error Message Editing

Privilege class G

Execution of DIAGNOSE Code X'5C' causes the editing of an error message according to the user's setting of the EMSG function.

***Entry Values***: Rx contains the address of the message to be edited. Ry contains the length of the message to be edited.

***Exit Values***: DMKHVC tests the VMMLEVEL field of the VMBLOK and returns to the caller with Rx and Ry modified as follows:

| VMMLEVEL | | Registers on Return | |
|---|---|---|---|
| VMMCODE | VMMTEXT | Rx | Ry |
| ON | ON | no change | no change |
| ON | OFF | no change | 10 (length of code) |
| OFF | ON | pointer to text part of message | length of text alone |
| OFF | OFF | N/A | 0 |

**Note:** DIAGNOSE Code X'5C' does not write the message; it merely rearranges the starting pointer and length. For CMS error messages, a console write is performed following the DIAGNOSE unless Ry is returned with a value of 0.

## DIAGNOSE Code X'60' - Determining the Virtual Machine Storage Size

Privilege class G

Execution of DIAGNOSE Code X'60' allows a virtual machine to determine its size. On return, the register specified as Rx contains the virtual machine storage size.

## DIAGNOSE Code X'64' - Finding, Loading, and Purging a Named Segment

Privilege class G

Execution of DIAGNOSE Code X'64' controls the linkage of discontiguous saved segments.

***Entry Values***: The type of linkage that is performed depends upon the function subcode in the register specified as Ry.

| Subcode | Function |
|---|---|
| X'00' | LOADSYS -- Loads a named segment in shared mode |
| X'04' | LOADSYS -- Loads a named segment in nonshared mode |
| X'08' | PURGESYS -- Releases the named segment from virtual storage |
| X'OC' | FINDSYS -- Finds the starting address of the named segment |

The register specified as Rx must contain the address of the name of the segment. The segment name must be 8 bytes long, on a doubleword boundary, left justified, and padded with blanks.

### The LOADSYS Function

When the LOADSYS diagnose function is executed, CP finds the system name table entry for the segment and builds the necessary page and swap tables (two sets one for each processor, when running in attached processor mode). CP releases all the virtual pages of storage that are to contain the named segment and then loads the segment in those virtual pages. When the LOADSYS function is executed, CP expands the virtual machine size dynamically, if necessary. CP also expands the segment tables to match any expansion of virtual storage.

When LOADSYS executes successfully, the address of where the named segment was loaded is returned in the register specified as Rx. When the LOADSYS function loads a segment in shared mode, it resets instruction and branch tracing if either was active.

After a LOADSYS function executes, the storage occupied by the named segment is addressable by the virtual machine, even if that storage is beyond the storage defined for the virtual machine. However, any storage beyond that defined for the virtual machine and below that defined for the named segment is not addressable. Figure 32 shows the virtual storage that is addressable before and after the LOADSYS function executes.

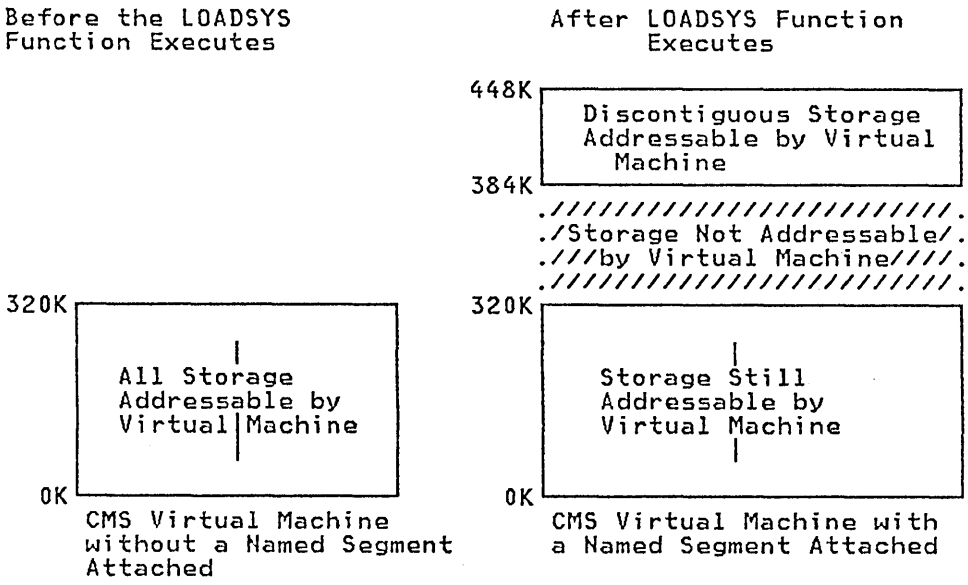



**Figure 32. Addressable Storage Before and After a LOADSYS Function**

When you save a named segment that is later loaded by the LOADSYS function, you must be sure that the addresses at which segments are saved are correct and that they do not overlay required areas of storage in the virtual machine. This is crucial because the LOADSYS function invokes the PURGESYS function before it builds the new page and swap tables. CP purges all saved systems that are overlayed in any way by the saved system it is loading.

***Condition Codes***: A condition code of 0 in the PSW indicates that the named segment was loaded successfully; the Rx register contains the load address.

A condition code of 1 in the PSW indicates the named segment was loaded successfully within the defined storage of the virtual machine. The Rx register contains the address at which the named segment was loaded. The Ry register contains the ending address of the storage released before the named segment was loaded.

**Note:** CMS only allows named segments to be attached beyond the defined size of the virtual machine. A condition code of 2 in the PSW indicates the LOADSYS function did not execute successfully. Examine the return code in the Ry register to determine the cause of the error.

| Return Code | Meaning |
|---|---|
| 44 | Named segment does not exist |
| 177 | Paging I/O errors |

### The PURGESYS Function

When the PURGESYS function is executed; CP releases the storage, and associated page and swap tables, that were acquired when the corresponding LOADSYS function was executed. If the storage occupied by the named segment was beyond the defined virtual machine storage size, that storage is no longer addressable by the virtual machine.

When a PURGESYS function is executed for a segment that was loaded in non-shared mode, the storage area is cleared to binary zeros. If PURGESYS is invoked for a named segment that was not previously loaded via LOADSYS, the request is ignored.

***Condition Codes***: A condition code of 0 in the PSW indicates successful completion.

A condition code of 1 in the PSW indicates that the named segment was not found in the virtual machine.

A condition code of 2 in the PSW and a return code of 44 in the Ry register indicate that the named segment either does not exist or was not previously loaded via the LOADSYS function.

### The FINDSYS Function

When the FINDSYS function is executed, CP checks that the named segment exists and that it has not been loaded previously.

***Condition Codes***: A condition code of 0 in the PSW indicates that the named segment is already loaded. The address at which it was loaded is returned in the register specified as Rx and its highest address is returned in the Ry register.

A condition code of 1 in the PSW indicates that the named segment exists but has not been loaded. In this case, the address at which the named segment is to be loaded is returned in the register specified as Rx and the highest address of the named segment is returned in the Ry register.

A condition code of 2 in the PSW indicates the FINDSYS function did not execute successfully. Examine the return code in the Ry register to determine the error that occurred.

| Return Code | Meaning |
|---|---|
| 44 | Named segment does not exist |
| 177 | Paging I/O errors |

## DIAGNOSE Code X'68' -- Virtual Machine Communication Facility (VMCF)

Privilege class G

The DIAGNOSE Code X'68' is used by a virtual machine to initiate a subfunction of the Virtual Machine Communication Facility (VMCF).

***Entry Values***: The general register specified as Rx contains the virtual address, doubleword aligned, of a parameter list (VMCPARM). One of the entries in this parameter list is a subfunction code, specifying the particular request being initiated. The subfunctions and their codes are:

| Subfunction | Code |
|---|---|
| AUTHORIZE | X'0000' |
| UNAUTHORIZE | X'0001' |
| SEND | X'0002' |
| SEND/RECV | X'0003' |
| SENDX | X'0004' |
| RECEIVE | X'0005' |
| CANCEL | X'0006' |
| REPLY | X'0007' |
| QUIESCE | X'0008' |
| RESUME | X'0009' |
| IDENTIFY | X'000A' |
| REJECT | X'000B' |

A description of all the fields of the VMCPARM is contained in the section "Virtual Machine Communication Facility."

The general purpose register specified as the Rx register contains the address of the VMCPARM list.

The general register specified as Ry contains the return code upon completion of DIAGNOSE X'68' or the detection of an error condition. The return codes are contained in the section "Virtual Machine Communication Facility."

Rx and Ry can be any general register, R0 through R15. They may also be the same register.

## DIAGNOSE Code X'6C' -- Special Diagnose for Shadow Table Maintenance

Privilege class G

DIAGNOSE Code X'6C' is an internal DIAGNOSE instruction issued by MVS to VM/SP that is used only to pass the virtual address of a page table entry that maps to real page zero for the low storage protection facility.

***Entry Values***: The virtual address passed in the Rx register is stored in the EXTVPORL field of the ECBLOK. The VMVPOREL flag in the VMBLOK is set on. The Ry register is not used.

***Usage***: The DIAGNOSE X'6C' information is used when a V=R user issues a SET STBYPASS VR command, and also when a V=V user sets a high-water mark.

***Condition Codes***: If a BC mode virtual machine attempts to issue a DIAGNOSE X'6C' a condition code of 3 is reflected in the BC mode PSW.

## DIAGNOSE Code X'70' -- Activating the Time-of-Day (TOD) Clock Accounting Interface

Privilege class G

Diagnose Code X'70' enables an operating system that is running in a virtual machine to request timing information from CP. Each time the virtual machine is dispatched, CP provides the accumulated processor time the virtual machine has used and the TOD the virtual machine was dispatched. Programs that are running in the virtual machine may use the timing information to calculate the amount of processor time used by each job, by each job step, and so forth.

DIAGNOSE Code X'70' should be used by operating systems that use the store clock (STCK) instruction to obtain the TOD in order to calculate processor usage. Because there is no virtual TOD clock, calculations that use multiple STCK instructions may not reflect the time used by just one virtual machine. They may also include the time used by all virtual machines and by CP.

A virtual machine should issue DIAGNOSE Code X'70' only one time. Once issued, it is effective until the virtual machine is reset.

***Entry Values***: When DIAGNOSE Code X'70' is issued, the Rx register must contain the address of a 16-byte area, the communication area. The Ry register is not used.

The communication area must be aligned on a doubleword boundary and must be in the virtual machine's real storage, preferably in page zero. Page zero is preferred because CP always locks page zero and must also lock the page that contains the communication area. Thus, when page zero is used, CP does not have to lock an additional page.

***Usage***: After DIAGNOSE Code X'70' is issued, CP updates the communication area each time the virtual machine is dispatched. The first eight bytes of the communication area contain the total processor time the virtual machine has used. The last eight bytes contain the TOD CP last dispatched the virtual machine. Programs running in the virtual machine should not alter the communication area.

To use the information that CP has stored in the communication area, perform the following steps:

1. Obtain the current TOD by issuing the STCK instruction.

2. Compute the difference between the TOD obtained in step 1 and the TOD stored in the communication area. This difference is the amount of processor time the virtual machine has used since it was last dispatched.

3. To calculate the total amount of processor time the virtual machine has used up to the present time, add the processor time that is stored in the communication area to the difference obtained in step 2.

4. Ensure that the TOD value stored in the communication area has not changed since step 2 was performed. If it has changed, repeat the procedure from step 1.

***Specification Exception***: CP returns a specification exception if DIAGNOSE Code X'70' is issued and:

- The virtual machine does not have the ECMODE option.
- The communication area is not aligned on a doubleword boundary.
- The address in the Rx register is not within the virtual machine's real address range.
- DIAGNOSE Code X'70' has already been issued for the virtual machine.

## DIAGNOSE Code X'74' -- Saving or Loading a 3800 Named System

Privilege class A, B, or C

DIAGNOSE Code X'74' is invoked to save an image library as a 3800 named system or to load a named system into virtual storage when that named system is required by the 3800 printer.

***Entry Values***: When the DIAGNOSE Code X'74' is invoked, the Rx, Rx+1, Ry, and Ry+1 registers must contain the following:

- Registers Rx and Rx+1 - must contain the eight-character name of the system to be saved or loaded, left-justified and padded with blanks.
- Register Ry - must contain the virtual address at which to start saving or loading the named system.
- Register Ry+1 - must contain a X'00' in the high order byte if a LOAD operation is required, and a X'04' for a SAVE operation. The remainder of the register must contain the number of bytes to be saved or loaded into virtual storage.

***Error Conditions***: A specification exception occurs if Register 15 is specified in either Rx or Ry, or if the virtual address specified in Ry is not on a page boundary. If the area to be saved or loaded extends beyond the user's virtual storage, an addressing exception occurs. Finally, a privileged operation exception results if the user does not have privileged class A, B, or C. These exceptions cause abnormal termination (abend) and the user is notified.

*Condition Codes*: When DIAGNOSE Code X'74' processing completes, one of the following condition codes is placed into register Ry and returned to CP:

| Return Code | Meaning |
|---|---|
| X'00' | Load/save successfully performed |
| X'04' | Named system not found |
| X'08' | Named system currently active |
| X'0C' | Volid for system not CP owned |
| X'10' | Volid for system not mounted |
| X'14' | Too many bytes to load/save; residual byte count is in Ry+1 |
| X'18' | Paging error during load/save |
| X'1C' | Too few bytes to LOAD/SAVE. Needed byte count is in Ry+1. |

## DIAGNOSE Code X'78' -- MSS Communication

Privilege class G

DIAGNOSE Code X'78' is used to communicate with the VM/SP control program about MSS volume mounts and demounts.

*Entry Values*: The Ry register contains a subfunction code. The valid subfunction codes and their meanings are:

- X'00' - The virtual machine issuing the DIAGNOSE instruction is running OS/VS with MSS support and the DMKMSS program for MSS communication. The Rx register contains the device address of the virtual machine's MSS communicator virtual device.
- X'04' - The virtual machine is ready to process an MSS request. The MSSCOM block representing the request should be placed at the virtual machine address indicated by the Rx register.
- X'08' - An MSS request represented by the MSSCOM block located at the virtual machine address indicated by the Rx register has been accepted by the MSC.
- X'0C' - An MSS request represented by the MSSCOM block located at the virtual machine address indicated by the Rx register has been rejected by the MSC.
- X'10' - The DMKMSS program is no longer available to process MSS requests.
- X'14' - The DMKMSS program has created a list of all VUAs associated with this processor (cpuid) and requests CP to build its shared and non-shared SDG tables from that list.

*Error Conditions*: If the DIAGNOSE Code X'78' is specified incorrectly, CP terminates the user program with one of the following exceptions:

**Error Return (DMKHVC)**

| | |
|---|---|
| Protection Exception | No DMKSSS module exists |
| Specification Exception | MSSCOM crosses a page boundary |

DIAGNOSE Code X'78' condition codes and return codes are:

| | |
|---|---|
| CONDITION CODE=0 | Successful completion. |
| CONDITION CODE=1(DMKSSS) | Error Condition. Register 15 contains one of the following: |
| RC=4 | Subfunction code was either less than zero or greater than 16. |
| RC=8 | Subfunction code was within the valid range, but not a multiple of 4. |
| RC=12 | Addressing exception trying to bring in the buffer page. |
| RC=16 | Issuer is not the issuer of subfunction zero. |

## DIAGNOSE Code X'7C' -- Logical Device Support Facility

Privilege class G

DIAGNOSE Code X'7C' allows an application running in a virtual machine to drive a logical 3270 as if it were a real display station locally attached to the VM/SP system. Communication between the application and the logical device is done via the DIAGNOSE interface and a new external interrupt code.

***Entry Values***: Rx is a user-specified register (*not GR15*) containing the logical device number that is used to coordinate CP and local system operations. It is not used for the INITIATE function.

Rx+1, for the INITIATE function, this register contains in the low-order three bytes the device model, class, and type for the logical device to be created. For example, a 3277 Model 2 is represented as X'00024004'. Valid device types are: 3277 Model 2, and 3278 Models 2, 3, 4, and 5.

For the ACCEPT function, this is a register that contains the address of a data buffer.

For the PRESENT function, this is a register that contains either an address or a complemented address. If an address, it is the address of a single buffer of data 4096 bytes or less in length. If a complemented address, it is the address of a list that describes a data stream occupying multiple data buffers and/or greater than 4096 bytes in length.

Ry is a user-specified register (*not GR15*) containing the code of the logical device subfunction to be executed:

X'0001' ..... INITIATE
X'0002' ..... ACCEPT
X'0003' ..... PRESENT
X'0004' ..... TERMINATE
X'0005' ..... TERMINATE (all)

On completion of an ACCEPT function, this register contains the length of the data.

Ry+1
For the ACCEPT and PRESENT functions, this is a register that contains the length of the data buffer when Rx+1 specifies a buffer address. On completion of any DIAGNOSE operation, this register contains the return code.

***Completion and Condition Codes***: Return codes are returned from this facility in register Ry+1. PSW completion codes and return codes are described below. Subfunctions that apply specifically to given combinations of completion and return codes are shown in parentheses.

CC=0 Subfunction completed with no errors.

RC=0 (any) Normal completion.

RC=1 (ACCEPT) Indicates another ACCEPT required for another data stream.

RC=2 (ACCEPT) Indicates another ACCEPT required for next segment of current data stream.

CC=1 Error condition.

RC=1 (any) Invalid function used in register Ry.

RC=2 (ACCEPT) No data available.

RC=3 (ACCEPT) Buffer too short. No data transferred. Another ACCEPT is required to retrieve the data. Register Ry contains the required data length.

RC=4 (ACCEPT or PRESENT) One of the following:

- Buffer is greater than 4096 bytes.
- Buffer length is not positive.
- Buffer not in user's address space.
- Paging I/O error.

RC=5 (INITIATE) Already have eight virtual machines that have created logical devices. Logical devices can be created for a maximum of eight concurrently active virtual machines in a VM/SP system.

RC=9 (INITIATE) Max of 512 logical devices per virtual machine reached.

RC=10 (ACCEPT or PRESENT) FETCH or STORE protection violation.

CC=2 Busy condition

RC=1 (PRESENT) CP has pending data that must be accepted first. The PRESENT is not performed.

RC=2 (PRESENT) A previous PRESENT has not completed execution. The current PRESENT is not performed. An external interrupt is issued to indicate when this PRESENT should be reissued.

RC=3 (PRESENT) CP has an active READ BUFFER command. The PRESENT issued is for READ MODIFIED data.

RC=4 (PRESENT) The data presented is from a READ BUFFER. No CP READ is outstanding, or the READ is a READ MODIFIED.

CC=3

a. (INITIATE) Logical device type, class, or model is invalid.

b. (Other functions) Logical device number in register Rx is invalid.

RC=1 (ACCEPT, PRESENT, TERMINATE) CP is in the process of terminating the logical device.

RC=2 (ACCEPT, PRESENT, TERMINATE) The logical device number does not exist.

RC=3 (INITIATE) The logical device class, type, or model is invalid.

### *Descriptions of Logical Device Support Facility Subfunctions*

Logical device subfunctions manage communications and the transfer of data between CP and the virtual machine for which the logical device was created.

INITIATE: DIAGNOSE CODE X'7C' SUBFUNCTION CODE X'0001'

The INITIATE subfunction opens a logical communications path between the calling virtual machine issuing the DIAGNOSE and the VM/SP Control Program. It causes a logical device to be created and the VM/370 logo to be directed to it. This results in an external interrupt to the issuing virtual machine to indicate that CP has data to be processed.

Register Rx+1 must contain the model number in byte 1, and the device class and type in bytes 2 and 3. Register Rx is not used for input.

The address of the logical device is placed in register Rx. This value is used on subsequent DIAGNOSE operations to indicate the logical device being used. This address is also provided with the external interrupt so that the issuing virtual machine can associate the interrupt with a specific logical device.

ACCEPT: DIAGNOSE CODE X'7C' SUBFUNCTION CODE X'0002'

The ACCEPT subfunction reads data that CP has directed to a logical device. It is invoked after the virtual machine that created the logical device is notified via external interrupt that output data is to be processed. Upon invocation register Rx+1 must contain the data buffer address and register Ry+1 the buffer length. If the data buffer supplied was too short to contain the data, the subfunction returns the required buffer size in register Ry and no data is moved. This action can be overridden by setting an indicator in the length register (bit zero in Ry+1 set to 1) when the subfunction is invoked. In this case, the data *is* be moved to the short buffer and a CC=0, RC=2 is sent. The system moves the next portion of the data on the next ACCEPT. Upon successful completion of subfunction processing, the data length is returned in Ry, and the data buffer contains the CCW OP code in its first byte and data in the remaining buffer space.

PRESENT: DIAGNOSE CODE X'7C' SUBFUNCTION CODE X'0003'

The PRESENT subfunction passes input data to CP. The location of the data is described by an address or a complemented address in register Rx+1. If the register contains an address, it is the address of a data buffer 4096 bytes or less in length. In this case, register Ry+1 contains the length of that data buffer. If register Rx+1 contains a complemented address, it is the address of a list that describes a data stream occupying multiple data buffers and/or greater than 4096 bytes in length. In this case, register Ry+1 is not used to describe the data length. However, in either case, a high-order bit of 1 in register Ry+1 indicates that the response is to a READ BUFFER command. Data format is the same as that produced by a local display control unit in response to a READ MODIFIED channel command.

If a list is used to describe the data, the list must be in the format:

| | |
|---|---|
| Length SEG1 | Address SEG1 |
| Length SEG2 | Address SEG2 |
| . | . . |
| . | . . |
| Length SEGn* | Address SEGn |

*Last entry indicated by a 1 in bit zero of its length field.

The list must start on a fullword boundary. Each entry consists of two fullword fields that describe the length and location of sequential segments of a data stream. A single entry list may be used to describe a single data buffer greater than 4096 bytes in length. Neither the list nor the data may be modified before transfer of the data has completed. An external interrupt signals completion of data transfer.

TERMINATE: DIAGNOSE CODE X'7C' SUBFUNCTION X'0004'

The TERMINATE subfunction notifies CP to drop a specific logical device. If the logical device is the console of a virtual machine, the virtual machine is placed in FORCE DISCONNECT state. If the logical device is DIALed to a virtual machine, it is detached from that virtual machine. If an input or output operation is being processed, it is terminated with a unit check and intervention required.

TERMINATE (ALL): DIAGNOSE CODE X'7C' SUBFUNCTION X'0005'

The TERMINATE (all) subfunction notifies CP to terminate all logical devices created for the issuing virtual machine.

## *External Interrupt Code X'2402'*

The logical device support uses a service signal interrupt, (class 24 external interrupt) to notify the virtual machine of a change in status for a specific logical device. The external interrupt code is X'2402'. This interrupt causes a full word of data to be stored at location 128 (decimal) in the virtual machine. The interrupt is masked on and off by bit 22 of control register 0.

The format of the stored fullword is:

128-129 logical device address

130 flag byte
bit zero - PRESENT subfunction purged
bit one - error in transmission or list

131 interrupt reason code

The logical device address is returned to the user after an INITIATE, and must be specified by the user for an ACCEPT, PRESENT, or TERMINATE.

Flag byte, bit zero is set to 1 if the data from the last PRESENT has been discarded by the system (subsequent I/O to the logical device was a WRITE instead of a READ). Flag byte, bit one is set to 1 if an error was encountered in the address list describing multiple buffers of a data stream, or one of the specified addresses in the list was not accessible. Otherwise, the flag byte remains zero.

The reason codes are:

01 - CP is terminating the connection

02 - A WRITE has been issued, so an ACCEPT must be done. (External interrupt flag byte, bit zero also indicates whether previous PRESENT data has been discarded.)

03 - A previous PRESENT is now finished (user received CC=2 and RC=2 after a PRESENT).
- A PRESENT has been suspended because of a transmission error. (External interrupt flag byte, bit one indicates this.)

04 - A READ BUFFER has been issued.

05 - A READ MODIFIED has been issued.

Reason code 1 indicates that the logical device no longer exists. The user receives a condition code 3 if he attempts to perform another function with this device.

## *Logical Device Restrictions*

The only devices supported are the local 3277 Model 2, and the local 3278 Models 2, 3, 4, and 5.

## DIAGNOSE Code X'80' -- MSSFCALL

Privilege class G

DIAGNOSE code X'80' is the VM/SP interface for communicating between CP and the Monitoring and Service Support Facility (MSSF). MSSF is a hardware component of the processor controller of the 3081 processor complex; it provides system configuration and storage information for the 3081 processor complex.

***Entry Values***: Rx is a user-specified register that contains the address of the MSSF data block (MSFBLOK). MSFBLOK is defined in increments of 8 bytes to a maximum of 2048 bytes. It must be aligned on a 2K boundary. MSFBLOK is locked in storage during the MSSFCALL request.

Ry is a user-specified register that contains the MSSF command word representing the function that MSSF is to perform. (See MSSF command words below.)

***Usage***: The CP module DMKMHC issues a real DIAGNOSE X'80' and services all MSSF external interruptions. DMKMHC issues a real DIAGNOSE X'80' when:

1. The operator issues a VARY ONLINE PROCESSOR nn command or a VARY OFFLINE PROCESSOR nn VPHY command. (These commands modify the real processor configuration to bring the processor physically online or offline.)

2. A V=R virtual machine running in EC mode issues the MSSF command word SCPINFO.[6] (Operating systems running in a virtual machine use the MSSF SCPINFO command to get information about a system configuration and storage allocation.

3. A user with privilege class C or E runs the IOCP program to read from or write to the Input/Output Configuration Data Set (IOCDS).

When CP ISSUES DIAGNOSE X'80', the hardware call block that is created (HCBLOK) contains the MSFBLOK address, the MSSF command word, and the address to return to after the MSSF has processed the request. See *VM/SP Data Areas and Control Block Logic, Volume 1* for the format of the MSFBLOK and HCBLOK.

### *MSSF COMMAND WORDS*

| | |
|---|---|
| X'0011nn01' | VARY ONLINE PROCESSOR nn where nn is the ID of the processor to be varied online. CP use only. |
| X'0010nn01' | VARY OFFLINE PROCESSOR nn VPHY where nn is the ID of the processor to be physically varied offline. CP use only. |
| X'00020001' | SCPINFO command. Virtual machine use only. |

[6] When a V=V virtual machine issues SCPINFO, DMKMHV does not pass control to DMKMHC. CP does not issue the DIAGNOSE X'80' but simulates the MSSF response and returns predefined data and status codes to the user. See *VM/SP System Logic and Problem Determination Guide, Volume 1 - CP* for a description of the pre-defined data.

X'00400002' IOCP special command to obtain or release the IOCP write lock.

X'00400102' IOCP WRITE to Level 1 IOCDS.

X'00410002' IOCP READ from Level 0 IOCDS.

X'00410102' IOCP READ from Level 1 IOCDS.

***Completion and Condition Codes***: Two possible condition codes returned for DIAGNOSE X'80' are:

cc=0 MSSF is processing the MSSFCALL request.

cc=2 MSSF is busy.

**Note:** If CP issued the MSSFCALL request, and MSSF was already processing a previous MSSFCALL request, abend MHC001 occurs. If a V=R user issued the request, CP reflects the condition code (2) to the virtual machine's PSW.

At the completion of an MSSFCALL, the following actions occur:

- MSSF generates a service signal interrupt, external interrupt X'2401' (class 24 external interrupt). This interrupt stores the absolute address of the MSSF data block (MSFBLOK) at decimal locations 128-131 in the virtual machine's PSA. Bit 22 of control register 0 controls masking of the service signal.
- MSSF passes a completion status code back in the MSFBLOK.
- CP returns control to the address specified in HCBLOK.
- If tracing is on, trace table entry X'17' traces all MSSFCALL requests. Refer to "Figure 8. CP Trace Table Entries" for the format and content of the entry. In addition, trace table entry X'01' reflects external interrupt X'2401' when the MSSF generates the service signal interrupt to CP.

Successful MSSF completion status codes are:

0010 - SCPINFO complete.

0020 - The processor is varied online/offline.

0120 - The processor is already varied online/offline. IOCP operation complete.

8020 - An IOCP READ is invalid because the file is open for writing.

4020 - The IOCP read or write operation was performed on the active IOCDS.

2020 - The active IOCDS has been written to.

Reject status codes are:

01F0 - Invalid command code or identification byte.

41F0 - Attempt to read closed IOCDS.

0100 - Data block not aligned on a 2K boundary.

0200 - Data block length not a multiple of 8.

0300 - The data block length is not adequate for the amount of information requested.

## DIAGNOSE Code X'84' -- Directory Update-In-Place

Privilege class B

DIAGNOSE Code X'84' enables a class B user to replace certain data in any entry of the VM/SP directory. The user must specify the directory entry and may replace the following data:

- Logon password
- Virtual machine storage size
- Maximum virtual machine storage size
- Privilege classes
- Dispatching priority
- Logical editing symbols
- Initial program load (IPL) system
- IPL parameter data
- Account number
- Distribution word
- User options
- Minidisk access mode
- Minidisk read, write, or multiple password
- Options of the SCREEN directory control statement

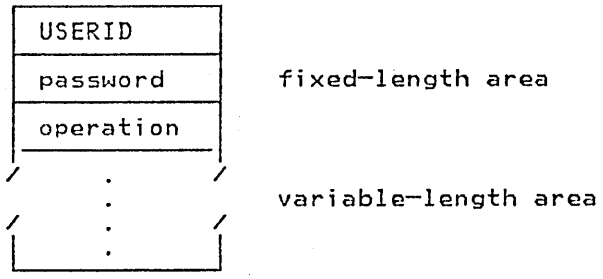
With the exception of the account number, all changes to the entry take effect the next time the USERID associated with the entry logs onto VM/SP. The account number may be updated such that the change (1) takes effect immediately, (2) takes effect immediately but is temporary lasting only until the USERID is logged off, or (3) takes effect the next time the USERID associated with the entry is logged on.

DIAGNOSE Code X'84' cannot add new entries to the directory, cannot delete existing entries, nor can it alter directory user-description statements. It can only replace existing directory data. Data is replaced in the form of the directory created by the directory service program, that is, in VM/SP control blocks.

For a detailed description of the directory data, see the *VM/SP Planning Guide and Reference*.

***Entry Values***: When DIAGNOSE Code X'84' is issued, the Rx register must point to a variable length parameter list and the Ry register must specify, in bytes, the length of the list. The list cannot be greater than 112 bytes long or less than zero bytes. The parameter list contains an area of fixed length followed by an area of variable length. Data in the fixed-length area identifies the directory entry to be updated, the password of the USERID associated with the entry, and the data field to be replaced in the directory entry. The variable-length area contains replacement data for the directory entry. All entries in the parameter list must contain unpacked, EBCDIC data.

The parameter list is organized as follows:




**Fixed-length area**

USERID

The USERID of the user whose directory entry is updated. This is an eight-character, left-justified value and must be padded with blanks.

password

The current password of the USERID whose directory entry is updated. This is an eight-character, left-justified value and must be padded with blanks.

If this field is blank, the update-in-place function is processed in 'testmode'. When the DIAGNOSE is issued in this fashion, the directory is not updated. 'Testmode' lets you check the syntax of the directory statements without accessing the directory disk.

operation

An eight-byte, left-justified character string that identifies the data in the directory entry that is to be replaced. Valid values and the data that each identifies for replacement are defined in the description of the variable-length area which follows.

**Variable-length area**

The following diagram shows for each value of the operation field, (1) the data that must be in the variable-length area of the parameter list, and (2) the format and characteristics of the data.

| **Operation Field Value** | **Date** | **Characteristics/Format** |
|---|---|---|
| LOGPASS | logon password | An eight-byte, left-justified value padded with blanks. |
| STORAGE | virtual machine storage size | An eight-byte, left-justified decimal value followed by the letter K. Pad with blanks following the letter K. |

| Operation Field Value | Date | Characteristics/Format |
|---|---|---|
| MAXSTOR | maximum virtual machine storage size | An eight-byte, left-justified decimal value followed by the letter K. Pad with blanks following the letter K. |
| PRIVLEGE | privilege classes | An eight-byte value where each byte represents a privilege class. Valid values for each byte are A through H. All existing classes in the directory entry are replaced. Therefore, specify existing classes that are to be retained as well as classes that are to be changed. The data must be left-justified and padded with blanks. |
| PRIORITY | dispatching priority | An eight-byte, left-justified value where the first two bytes, counting from the left, specify the dispatching priority. Valid values for these bytes are 1 - 99. Values 1 through 9 must be padded with a blank. The other six bytes are reserved for IBM use. |
| EDITCHAR | logical editing symbols | An eight-byte value where the first four bytes, counting from the left, are line edit symbols. The first or high-order byte is the "line-end" symbol, the second byte is the "line-delete" symbol, the third byte is the "character-delete" symbol, and the fourth byte is the "escape-character" symbol. All existing symbols in the directory are replaced. Therefore, specify existing symbols that are to be retained as well as symbols that are to be changed. Unspecified symbols must contain blanks. The last four bytes of the eight-byte value are reserved for IBM use. |
| IPL | system name or virtual device address and variable data | A one-to-eight character value, left-justified and padded with blanks, followed by the keyword PARM and up to 48 characters of variable data. All existing values are replaced in the directory entry; therefore, specify values that are to be retained as well as values that are to be changed. Trailing blanks are not truncated but passed. |

| Operation Field Value | Date | Characteristics/Format |
|---|---|---|
| ACCOUNT | account number | A one-to-eight character value, left justified and padded with blanks. (This change takes effect the next time the USERID is logged on.) |
| IACCOUNT | account number | A one-to-eight character value left-justified and padded with blanks. (This change takes effect immediately.) |
| TACCOUNT | account number | A one-to-eight character value, left-justified and padded with blanks. (This change takes effect immediately but is temporary, lasting only until the USERID is logged off.) |
| DISTRIB | distribution identification word | A one-to-eight character value, left-justified and padded with blanks. |
| OPTIONS | user options | An eighty-byte, left-justified value padded with blanks. Specify each option as a character string with a blank character between options. For a description of each option and a list of valid values, see *VM/SP Planning Guide and Reference.* All existing options are replaced in the directory entry. Therefore, specify existing options that are to be retained as well as options that are to be changed. The options field must be followed by the value X'FFFFFFFF'. |

| Operation Field Value | Date | Characteristics/Format |
|---|---|---|
| MDISK | minidisk address, access mode, read password, write password, and multiple password | A thirty-byte field defined as follows. All values must be left justified and padded with blanks. Valid values for the access mode and for passwords are defined in *VM/SP Planning Guide and Reference.*<br><br>Bytes 1-3, counting from the left, specify a minidisk address. This is the minidisk whose mode and passwords will be changed. The address must already exist in the directory entry.<br><br>Bytes 4-6 specify the access mode.<br><br>Bytes 7-14 specify the read password.<br><br>Bytes 15-22 specify the write password.<br><br>Bytes 23-30 specify the multiple password.<br><br>The access mode, the read password, the write password, and the multiple password are replaced in the directory entry. Therefore, specify existing values that are to be retained as well as values that are to be changed. |

| Operation Field Value | Date | Characteristics/Format |
|---|---|---|
| SCREEN | display screen options | An eighty-byte area composed of ten doubleword fields. The ten fields are paired into five sets corresponding to the five display areas of the screen. You must specify these areas in the following order:<br><br>• CP output<br>• VM output<br>• input redisplay<br>• input area<br>• status area.<br><br>Each of the five doubleword sets has a color field and an extended highlight field. (See the SCREEN option description in the *VM/SP Planning Guide and Reference* for the valid color and extended highlight values.) Within each doubleword set you must specify the color first followed by the extended highlight value. You must specify all fields, including those you don't want to change. All of the options you specify must also be left justified in their eight-byte field. |

Before control is returned to the virtual machine, DIAGNOSE Code X'84' sets a condition code and, if errors were detected, a return code in the Ry register. The condition codes and return codes are defined as follows:

| Condition Code | Meaning |
|---|---|
| 0 | The directory was successfully updated. |
| 1 | DIAGNOSE Code X'84' detected an error. The directory is unchanged. The return code defines the error. |

| Return Code | Meaning |
|---|---|
| 10,11 | An error occurred writing the directory to a direct access device. To update the directory, use the directory service program described in the *VM/SP Planning Guide and Reference.* |
| 20 thru 27, 90, 112, 113 | DIAGNOSE Code X'84' encountered a processing error. To update the directory, use the directory service program described in the *VM/SP Planning Guide and Reference.* |
| 26 | Specified minidisk address does not exist in directory entry, or specified userid does not have any devices defined in the directory entry. |
| 28 | The value in the OPERATION field of the parameter list is invalid. |

| | |
|---|---|
| 30 | The specified USERID could not be found. |
| 31 | The password specified in the fixed-length area of the parameter list does not match the current password of the USERID being updated. |
| 40,41 | The value specified for the virtual machine storage size or for the maximum virtual machine storage size is too large. The maximum allowable size is 16 megabytes. |
| 42, 43 | The value specified for the virtual machine storage size or for the maximum virtual machine storage size contains a syntax error or an invalid character. |
| 50, 51 | The specified privilege classes are invalid. |
| 52, 53 | The specified privilege classes contain a syntax error or an invalid character. |
| 60, 61, 62 | The specified priority contains a syntax error or an invalid character. |
| 63 | The priority value is too large. The maximum allowable value is 99. |
| 65, 66 | Parameter list size error; if return code = 65, the list exceeds 112 bytes; if return code = 66, the list size is less than zero bytes long. |
| 70 | A specified option is invalid. The invalid options are VMSAVE, STFIRST, 370E, and MAXCONN nnnnn. |
| 71 | The value X'FFFFFFFF' was not coded after the list of options. |
| 72 | The option value contains a syntax error or an invalid character. |
| 80 | The parameter list contains an invalid minidisk address. |
| 81 | The parameter list specifies an invalid access mode for a minidisk. |
| 82, 83 | The minidisk read, write, or multiple password specified in the parameter list requires a change in the size of the directory entry. |
| 91 | No attributes were found on the SCREEN command. |
| 92 | Invalid attributes were found on the SCREEN command. |

| | |
|---|---|
| 101 | The parameter list is too large. |
| 102 | The parameter list is less than 1. |
| 110 | No parameter data currently exists in the directory entry. |
| 111 | The parameter length is invalid. |

## DIAGNOSE Code X'8C' -- Access Certain Device Dependent Information

Privilege class G

DIAGNOSE X'8C' allows a virtual machine to obtain certain device-dependent information without issuing a WRITE STRUCTURE FIELD QUERY REPLY. DIAGNOSE X'8C' retrieves this information from the RDEVBLOK or NICBLOK and creates a diagnose interface to enable the virtual machine to access it. Because the information is obtained only at power-on time, if the characteristics of the terminal are altered dynamically, those changes are not reflected in the data returned by DIAGNOSE X'8C'. If a NETWORK ENABLE is issued to a device with advanced features and a NETWORK ATTACH is issued prior to turning on the device, the advanced features are non-operational. The data returned by DIAGNOSE X'8C' reflects this status.

**Note:** Devices for which the write structure field is not applicable return all zeroes from the DIAGNOSE X'8C'.

***Entry Values***: DIAGNOSE X'8C' is invoked as follows:

Rx is the address of user-provided data buffer
Ry is the length of user-provided data buffer
Rx+1 is the virtual device address.

***Specification Exception***: If the length specified is negative, the virtual machine receives a specification exception from CP. If the length specified is greater than 6, the user receives six bytes of information and the Ry contains the residual count (length specified minus 6).

The user receives a specification exception if

- The length specified is negative
- The virtual device address specified is invalid
- The buffer address is not on a doubleword boundary.

The user receives an addressing exception if

- An invalid buffer address is specified.

***Exit Values***: The data returned by DIAGNOSE X'8C' is in the following format:

| Byte 0 | Byte 1 | Byte 2 and 3 | Byte 4 and 5 |
|---|---|---|---|
| Flags | Number of partitions | Screen width in cells | Screen height in cells |

Flags
80 = ext color
40 = ext highlight
20 = Programmable Symbol Sets (PSS) available
01 = 14-bit addressing

Upon completion, Rx contains:

- A 4 if an I/O error occurs
- A 0 if the DIAGNOSE completes successfully.

# CP Conventions

## CP Coding Conventions

The following are coding conventions used by CP modules. This information should prove helpful if you debug, modify, or update CP.

1. FORMAT:

| Column | Contents |
|---|---|
| 1 | Labels |
| 10 | Op Code |
| 16 | Operands |
| 31, 36, 41, etc. | Comments (see Item 2) |

2. COMMENT:

   Approximately 75 percent of the source code contains comments. Sections of code performing distinct functions are separated from each other by a comment section.

3. CONSTANTS:

   Constants follow the executable code and precede the copy files and/or macros that contain DSECTs or system equates. Constants are defined in a section followed by a section containing initialized working storage, followed by working storage. Each of these sections is identified by a comment. Wherever possible for a module that is greater than a page, constants and working storage are within the same page in which they are referenced.

4. No program modifies its own instructions during execution.

5. No program uses its own unlabeled instructions as data.

6. REGISTER USAGE:

   For CP, in general

| Register | Use |
|---|---|
| 6 | RCHBLOK, VCHBLOK |
| 7 | RCUBLOK, VCUBLOK |
| 8 | RDEVBLOK, VDEVBLOK |
| 10 | IOBLOK |
| 11 | VMBLOK |
| 12 | Base register for modules called via SVC |
| 13 | SAVEAREA for modules called via SVC |
| 14 | Return linkage for modules called via BALR |
| 15 | Base address for modules called via BALR |

   For Virtual-to-Real address translation

| Register | Use |
|---|---|
| 1 | Virtual address |
| 2 | Real address |

7. When describing an area of storage in mainline code, a copy file, or a macro, DSECT is issued containing DS instructions.

8. Meaningful names are used instead of self-defining terms: for example, 5,X'02',C'I' to represent a quantity (absolute address, displacement, length, should be symbolic and defined by an equate (EQU) listing. For example:

```
VMSTATUS EQU X'02'
```

   To set a bit, use:

```
OI BYTE,BIT
```

   where BYTE = name of field, BIT is an EQU symbol.

   To reset a bit, use:

```
NI BYTE,255-BIT
```

   To set multiple bits, use:

```
OI BYTE,BIT1+BIT2
```

   etc. ...

   All registers are referred to as:

   R0, R1, ..., R15.

   All lengths of fields or control blocks are symbolic, that is, length of VMBLOK is:

```
VMBLOKSZ EQU *-VMBLOK
```

9. Avoid absolute relative addressing in branches and data references, (that is, location counter value (*) or symbolic label plus or minus a self-defining term used to form a displacement).

10. When using a single operation to reference multiple values, specify each value referenced, for example:

```
LM R2,R4,CONT SET R2=CON1
          SET R3=CON2
          SET R4=CON3
.
.
.
CON1 DC F'1'
CON2 DC F'2'
CON3 DC F'3'
```

11. Do not use PRINT NOGEN or PRINT OFF in source code.

12. MODULE NAMES:

    Control Section Names and External References are as follows:

    Control Section or Module Name

    The first three letters of the module name are the assigned component code.

    Example: DMK

    The next three letters of the module name identify the module and must be unique.

    Example: DSP

    This three-letter, unique module identifier is the label of the TITLE card.

    Each entry point or external reference must be prefixed by the six letter unique identifier of the module.

    Example: DMKDSPCH

13. TITLE CARD:

    DSP TITLE 'DMKDSP VM/SP DISPATCHER VERSION v LEVEL l'

14. ERROR MESSAGES:

    There should be no insertions into the message at execution time and the length of the message should be resolved by the assembler. If insertions must be made, the message must be assembled as several DC statements, and the insert positions must be individually labeled.

15. For all RX instructions use a comma (,) to specify the base register when indexing is not being used, that is:

    L R2,AB(,R4)

16. To determine whether you are executing in a virtual machine or in a real machine, issue the Store Processor ID (STIDP) instruction. If STIDP is issued from a virtual machine, the version number (the first byte of the CPUID field) returned will be X'FF'.

## CP Loadlist Requirements

The CP loadlist EXEC contains a list of CP modules used by the VMFLOAD procedures when punching the text decks that will make up the CP system. All modules following DMKCPE in the list are pageable CP modules. Each 4K page in this area may contain one or more modules. The module grouping is governed by the order in which they appear in the loadlist. An SPB[7] (Set Page Boundary) card, a loader control card placed in the text file, forces the loader to start this module at

7 A 12-2-9 multipunch must be in column 1 of an SPB card and the characters SPB in columns 2, 3, and 4 respectively.

the next higher 4K boundary. The loader automatically moves a module to the next higher 4K boundary if it cannot fit in with its predecessors on the load list. In this case a message is placed on the load map:

"SPB INSERTED"

as part of the line

"**EXTERNAL SYMBOL DICTIONARY FOR DMKXXX"

An SPB card is required only for the first module following DMKCPE. If more than one module is to be contained in a 4K page, only the first can be assembled with an SPB card. The second and subsequent modules for a multiple module 4K page must not contain SPB cards.

The position of two modules in the loadlist is critical. All modules following DMKCPE must be reenterable and must not contain any address constants referring to anything in the pageable CP area. DMKCKP must be the last module in the loadlist. It is also recommended that DMKPSA be the first module in the CP resident nucleus.

No change should be made to the sequence of modules in the resident or pageable portion of the loadlist.

## How to Add a Console Function to CP

Installations may add their own commands to their VM/SP system. First, code the module to handle the command processing. Follow the CP coding conventions outlined in an earlier section of this book.

Second, add an entry for the command in the CP DMKCFC module. DMKCFC has two entry points: one for logged-on users and another for nonlogged-on users. If the command is for logged-on users, be sure its entry is beyond the label COMNBEG1.

To place an entry for the command in the DMKCFC module, insert a line with the following format:

| | | |
|---|---|---|
| [label] | COMND | commandname,class,min,entrypt[,NCL=1] |

**where:**

commandname is a 1- to 8-character name.

class — is the command privilege class (up to four classes are allowed). 0 is coded for nonlogged-on user commands or when NCL=1.

min — is the number of characters allowed as the minimum truncation.

entrypt — is the entry point of the module you write to process the new command.

NCL=1 — is specified if the command is to be allowed before the user logs on. When NCL=1, the class is not checked.

After the above entry has been inserted in the DMKCFC module, reload DMKCFC as a pageable module ensuring that it does not cross a page boundary. You must also load your own module which may or may not be a resident module.

# Print Buffers and Forms Control

The Forms Control Buffer for the 3203, 3262, 3289 Model 4, and 4245, is exactly like the 3211 Forms Control Buffer. Please note that the FCB loaded in a virtual 3203, 3211, or 3262 should be compatible with the FCB loaded in the real counterpart. Otherwise, the results can be unpredictable. The 3203 and 3262 use the Universal Character Set (UCS) used by the 1403 Printer.

The 3203 and 3262 attach a 64-byte associative field to the end of the UCS to check, during print line buffer (PLB) loading, that each character loaded into the PLB for printing is also on the print train. The 3203 associative field is exactly like the 3211 associative field described in Figure 33 on page 287, with the exception that the addresses begin at 240. You also need an associative field when making use of of the UCC buffer. Refer to your printer's Components Description Manual for a detailed layout of the associative field.

Buffer images are supplied for the UCS (Universal Character Set) buffer, the UCSB (Universal Character Set Buffer), the FOB (Font Offset Buffer), and the FCB (Forms Control Buffer). The VM/SP-supplied buffer images are:

- UCS - for the 1403 and 3203 Printers

| Name | Meaning |
|---|---|
| AN | Normal AN arrangement |
| HN | Normal HN arrangement |
| PCAN | Preferred character set, AN |
| PCHN | Preferred character set, HN |
| QN | PL/I - 60 graphics |
| QNC | PL/I - 60 graphics |
| RN | FORTRAN, COBOL commercial |
| YN | High speed alphanumeric |
| TN | Text printing 120 graphics |
| PN | PL/I - 60 graphics |
| SN | Text printing 84 graphics |

- UCSB - for the 3211 Printer

| Name | Meaning |
|---|---|
| A11 | Standard Commercial |
| H11 | Standard Scientific |
| G11 | ASCII |
| P11 | PLI |
| T11 | Text Printing |

- UCSB - for the 3262 Printer

| Name | Meaning |
|---|---|
| P48 | 48 character print image |
| P52 | 52 character print image (Austria/Germany) |
| P63 | 63 character print image, optimized |
| P64 | 64 character print image |
| P96 | 96 character print image |
| P116 | 116 character print image (French/Canadian) |
| P128 | 128 character print image (Katakana) |

- FOB - for the 3289 Model 4 printer

| Name | Meaning |
|---|---|
| F48 | Font Offset Buffer for the 48-character print belt |
| F64 | Font Offset Buffer for the 64-character print belt |
| F94 | Font Offset Buffer for the 94-character print belt |
| F127 | Font Offset Buffer for the 127-character print belt |

- FCB - for the 3203, 3211, 3262, 3289 Model 4, and 4245 Printers

Two names are provided for an FCB image.

| Name | Meaning |
|---|---|
| FCB1 | Space 6 lines/inch<br>Length of page 66 lines |

| Line Represented | Channel Skip Specification |
|---|---|
| 1 | 1 |
| 3 | 2 |
| 5 | 3 |
| 7 | 4 |
| 9 | 5 |
| 11 | 6 |
| 13 | 7 |
| 15 | 8 |
| 19 | 10 |
| 21 | 11 |
| 23 | 12 |
| 64 | 9 |

| Name | Meaning |
|---|---|
| FCB8 | Space 8 lines/inch<br>Length of page 68 lines |

| Line Represented | Channel Skip Specification |
|---|---|
| 1 | 1 |
| 4 | 2 |
| 8 | 3 |
| 12 | 4 |
| 16 | 5 |
| 20 | 6 |
| 24 | 7 |
| 28 | 8 |
| 32 | 10 |
| 36 | 11 |
| 63 | 12 |
| 66 | 9 |

For the exact contents of the buffer images, see the

*IBM 2821 Control Unit Component Description*

*IBM 3211 Printer*

*IBM 3216 Interchangeable Train Cartridge*

*IBM 3811 Printer Control Unit Component Description and Operator's Guide*

*IBM 3289 Line Printer Model 4 and Component Description*

*IBM 3262 Printers I and II Component Description.*

The following table indicates in what module the images for each printer should be coded:

| Data Module | Printer |
|---|---|
| DMKFCB | All 3211 type printers |
| DMKUCS | UCS image for the 1403 printer |
| DMKUCC | UCS image for the 3203 printer |
| DMKPIA | UCS image for the 3289E printer |
| DMKPIB | UCS image for the 3262-1/11 printer |
| DMKUCB | UCS image for the 3211 printer |

For further information refer to the Component Description of the printer for which the image is being coded.

If you find that the supplied buffer images do not meet your needs, you can alter a buffer image or create a new buffer image. Be careful not to violate the VM/SP coding conventions if you add a new buffer image; buffer images must not cross page boundaries.

## Adding New Print Buffer Images

In order to add a new print buffer image to VM/SP, you must:

1. Provide a buffer image name and 12 byte header for the buffer load.
2. Provide the exact image of the print chain.
3. Provide a means to print the buffer image if VER is specified on the LOADBUF command.
4. Reload the changed CP modules.

Macros are available that make the process of adding buffer images relatively easy and should be used to avoid errors.

### *UCS Buffer Images for the 1403 Printer*

The Universal Character Set (UCS) buffer contains up to 240 characters and supports the 1403 printer. To add a new UCS buffer image, first code the UCS macro. This creates a 12-byte header for the buffer load that is used by CP. The format of the UCS macro is:

| | | |
|---|---|---|
| | UCS | ucsname |

**where:**

ucsname is a 1- to 4-character name that is assigned to the buffer load.

Next, supply the exact print image. The print image is supplied by coding DCs in hexadecimal or character format. The print image may consist of several DCs, the total length of the print image cannot exceed 240 characters.

The UCSCCW macro must immediately follow the print image. This macro creates a CCW string to print the buffer load image when VER is specified by the operator on the LOADBUF command. The format of the UCSCCW macro is:

| | | |
|---|---|---|
| | UCSCCW | ucsname[,(print1,print2,...,print12)] |

**where:**

ucsname is a 1- to 4-character name that is assigned to the buffer load by the UCS macro.

[(print1,...,print12)]
is the line length (or number of characters to be printed by the corresponding CCW) for the verify operation. Each count specified must be between 1 and 132 (the length of the print line on a 1403 printer) and the default line length is 48 characters. Up to 12 print fields may be specified. However, the total number of characters to be printed may not exceed 240.

Finally, insert the macros just coded, UCS and UCSCCW, into the DMKUCS module. This module must be reloaded. DMKUCS is a pageable module with no executable code. DMKUCS must be on a page boundary and cannot exceed a full page in size. If DMKUCS exceeds a page boundary (4K), an error message is issued.

### Examples of New UCS Buffer Images

**Example 1:** You do not have to specify the line length for verification of the buffer load. Insert the following code in DMKUCS:

```
UCS     EX01
DC      5CL'1234567890A...Z1234567890*/'
UCSCCW EX01
```

The buffer image is 5 representations of a 48-character string containing:

- The alphabetic characters
- The numeric digits, twice
- The special characters: * and /

Since the line length for the print verification is not specified on the UCSCCW macro, it defaults to 48 characters per line for 5 lines.

**Example 2:** Insert the following code in DMKUCS:

```
UCS     NUM1
DC      24CL'1234567890'
UCSCCW NUM1,(60,60,60,60)
```

The NUM1 print buffer consists of twenty-four 10-character entries. If, after DMKUCS is reloaded, the command

```
LOADBUF  00E  UCS  NUM1  VER
```

is specified, 4 lines of 60 characters (the 10-character string repeated 6 times) are printed to verify the buffer load).

**Example 3:** The print image can be specified in character or hexadecimal notation, or a combination of the two. The code in DMKUCS to support the preferred character set, AN, is as follows:

```
UCS     PCAN
DC      C'1234567890,-PQR
DC      C'.*1234567890,-JKLMNOABCDEFGHI+.*'
DC      C'1234567890,-PQR&&$%/STUVWXYZ','9C'
DC      C'.*1234567890,-JKLMNOABCDEFGHI+.*'
DC      C'1234567890,-PQR
DC      C'.*1234567890,-JKLMNOABCDEFGHI+.*'
DC      C'1234567890,-PQR&&$%/STUVWXYZ','9C'
DC      C'.*1234567890,-JKLMNOABCDEFGHI+.*'
UCSCCW PCAN,(60,60,60,60)
```

The DCs are coded in both character and hexadecimal notation. The hexadecimal code for the lozenge ('9C') follows the character notation on 4 of the DCs. The DCs, when taken in pairs, represent 60 characters. When print verification of a buffer load is requested, 4 lines of 60 characters are printed.

## *UCSB Buffer Images for the 3211 Printer*

The Universal Character Set Buffer (UCSB) contains up to 512 characters and supports the 3211 printer. To add a new UCSB image, first code the UCB macro. This macro creates a 12-byte header record for the buffer load that is used by CP. The format of the UCB macro is:

| | | |
|---|---|---|
| | UCB | ucbname |

**where:**

ucbname is a 1- to 4-character name that is assigned to the buffer load.

Next, supply the exact print image. The print image is supplied by coding DCs in hexadecimal or character notation. The total length of the print image cannot exceed 512 characters.

The format of the UCSB is:

| Position | Contents |
|---|---|
| 1-432 | Print train image. |
| 433-447 | Reserved for IBM use. Must be all zeros. |
| 448-511 | Associative field. See Figure 33 on page 287 for an explanation of the contents of this field. The associative field is used to check |

(during print line buffer (PLB) loading) that each character loaded into the PLB for printing also appears in the train image field of the UCSB and, therefore, is on the print train. Any character loaded into the PLB without its associated code in the train image field of the UCSB is nonprintable, and causes a "print data check" to be set immediately. The associative field also contains dual control bits.

512 Reserved for IBM use. Must be zero.

The UCBCCW macro must immediately follow the print image. This macro creates a CCW string to print the buffer load image when the operator specifies VER on the LOADBUF command. The format of the UCBCCW macro is:

| | | |
|---|---|---|
| | UCBCCW | ucbname[,(print1,print2,...print12)] |

**where:**

ucbname is 1- to 4-character name that is assigned to the buffer load by the UCB macro.

[(print1,...,print12)]
specifies the line length of each line (up to 12) printed to verify the buffer load. The line length must be between 1 and 150 (the length of a print line on a 3211 printer). The default specification for verification is 48 characters per line for nine lines. The total number of characters to be printed must not exceed the size of the print train image, 432 characters.

Finally, insert the two macros just coded, UCB and UCBCCW, into the DMKUCB module. This module must be reloaded before the new buffer image can be used. DMKUCB is a pageable module with no executable code. DMKUCB must be on a page boundary and cannot exceed a full page in size. If DMKUCB exceeds a page boundary (4K), an error message is issued.

### Examples of UCSB Images

The code for the A11 UCSB is as follows:

```
*       A11 STANDARD COMMERCIAL 48 GRAPHICS 3211
        UCB     A11
        DC      9C'1<.+IHGFEDCBA*$-RPQONMLKJ%,&&ZYXWVUTS/@098765432'
        DC      X'000000' 433-435
        DC      X'00000000000000000000000101010' 436-450
        DC      X'10101010101010004040424000 4010' 451-465
        DC      X'1010101010101010004040410000 40' 466-480
        DC      X'4010101010101010100040400000 000' 481-495
        DC      X'1010101010101010101000404044 48' 496-510
        DC      X'0000' 511-512
        UCBCCW A11,(48,48,48,48,48,48,48,48,48)
        EJECT
```

Note that the DC specification contains 49 characters and the UCBCCW macro specifies 48 characters. The ampersand (&) must be coded twice in order to be accepted by the assembler. The single quote (') must also be specified twice in order to be accepted.

It would have been acceptable to code the UCBCCW as:

since the default is what was coded.

| | Bit 0 | | Bit 1 | | Bit 2 | | Bit 3 | |
|---|---|---|---|---|---|---|---|---|
| UCSB Address | Hexa-decimal | Graphic & Control Symbols EBCDIC | Hexa-decimal | Graphic & Control Symbols EBCDIC | Hexa-decimal | Graphic & Control Symbols EBCDIC | Hexa-decimal | Graphic & Control Symbols EBCDIC |
| 448 | 00 | NUL | 40 | SP | 80 | | C0 | { |
| 449 | 01 | | 41 | | 81 | a | C1 | A |
| 450 | 02 | | 42 | | 82 | b | C2 | B |
| 451 | 03 | | 43 | | 83 | c | C3 | C |
| 452 | 04 | PF | 44 | | 84 | d | C4 | D |
| 453 | 05 | HT | 45 | | 85 | e | C5 | E |
| 454 | 06 | LC | 46 | | 86 | f | C6 | F |
| 455 | 07 | DEL | 47 | | 87 | g | C7 | G |
| 456 | 08 | | 48 | | 88 | h | C8 | H |
| 457 | 09 | | 49 | | 89 | i | C9 | I |
| 458 | 0A | | 4A | ¢ | 8A | « | CA | |
| 459 | 0B | | 4B | . | 8B | » | CB | |
| 460 | 0C | | 4C | < | 8C | ≤ | CC | ⌡ |
| 461 | 0D | | 4D | ( | 8D | ( | CD | |
| 462 | 0E | | 4E | + | 8E | + | CE | ⌠ |
| 463 | 0F | CU1 | 4F | \| | 8F | | CF | ) |
| 464 | 10 | | 50 | & | 90 | | D0 | } |
| 465 | 11 | | 51 | | 91 | j | D1 | J |
| 466 | 12 | | 52 | | 92 | k | D2 | K |
| 467 | 13 | | 53 | | 93 | l | D3 | L |
| 468 | 14 | RES | 54 | | 94 | m | D4 | M |
| 469 | 15 | NL | 55 | | 95 | n | D5 | N |
| 470 | 16 | BS | 56 | | 96 | o | D6 | O |
| 471 | 17 | IL | 57 | | 97 | p | D7 | P |
| 472 | 18 | | 58 | | 98 | q | D8 | Q |
| 473 | 19 | | 59 | | 99 | r | D9 | R |
| 474 | 1A | CC | 5A | ! | 9A | ª | DA | |
| 475 | 1B | | 5B | $ | 9B | º | DB | |
| 476 | 1C | | 5C | * | 9C | æ | DC | |
| 477 | 1D | | 5D | ) | 9D | ) | DD | |
| 478 | 1E | | 5E | ; | 9E | ± | DE | |
| 479 | 1F | CU2 | 5F | ¬ | 9F | Æ | DF | |
| 480 | 20 | | 60 | - | A0 | ¸ | E0 | \ |
| 481 | 21 | | 61 | / | A1 | ~ | E1 | |
| 482 | 22 | | 62 | | A2 | s | E2 | S |
| 483 | 23 | | 63 | | A3 | t | E3 | T |
| 484 | 24 | BYP | 64 | | A4 | u | E4 | U |
| 485 | 25 | LF | 65 | | A5 | v | E5 | V |
| 486 | 26 | EOB | 66 | | A6 | w | E6 | W |
| 487 | 27 | PRE | 67 | | A7 | x | E7 | X |
| 488 | 28 | | 68 | | A8 | y | E8 | Y |
| 489 | 29 | | 69 | | A9 | z | E9 | Z |
| 490 | 2A | SM | 6A | ¦ | AA | | EA | |
| 491 | 2B | | 6B | , | AB | ⌊ | EB | |
| 492 | 2C | | 6C | % | AC | ⌈ | EC | ⌐ |
| 493 | 2D | | 6D | > | AD | [ | ED | |
| 494 | 2E | | 6E | | AE | ≥ | EE | |
| 495 | 2F | CU3 | 6F | ? | AF | ° | EF | |
| 496 | 30 | | 70 | | B0 | 0 | F0 | 0 |
| 497 | 31 | | 71 | | B1 | 1 | F1 | 1 |
| 498 | 32 | | 72 | | B2 | 2 | F2 | 2 |
| 499 | 33 | | 73 | | B3 | 3 | F3 | 3 |
| 500 | 34 | PN | 74 | | B4 | 4 | F4 | 4 |
| 501 | 35 | RS | 75 | | B5 | 5 | F5 | 5 |
| 502 | 36 | UC | 76 | | B6 | 6 | F6 | 6 |
| 503 | 37 | EOT | 77 | | B7 | 7 | F7 | 7 |
| 504 | 38 | | 78 | | B8 | 8 | F8 | 8 |
| 505 | 39 | | 79 | \ | B9 | 9 | F9 | 9 |
| 506 | 3A | | 7A | ! | BA | | FA | |
| 507 | 3B | | 7B | # | BB | ⌋ | FB | |
| 508 | 3C | | 7C | @ | BC | ⌉ | FC | |
| 509 | 3D | | 7D | ' | BD | ] | FD | |
| 510 | 3E | | 7E | = | BE | ≠ | FE | |
| 511 | 3F | | 7F | " | BF | ± | FF | |

**Figure 33. UCSB Associative Field Chart**

## *FOB Buffer Images for the 3289 Model 4 Printer*

The Font Offset Buffer (FOB) contains 256 font offset bytes and supports the 3289 Model 4 printer. A font offset byte defines a character by specifying its location on a print belt. The location of the character is specified in terms of its distance (offset) from a fixed reference point.

To add a new FOB, create a header for the buffer, supply the contents of the new buffer, and provide a means to print the buffer image if the operator must verify its contents.

First, code the FOB macro instruction to create a 12-byte header record to be used by the CP. The format of the FOB macro instruction is:

| | | |
|---|---|---|
| | FOB | fobname |

**where:**

fobname is a 1- to 4-character name that is assigned to the buffer.

Next supply the exact contents of the Font Offset Buffer by coding DCs in hexadecimal format. Several DCs can be coded, but the total buffer length must be 256 bytes. (If the buffer does not contain 256 bytes, the load check bit is set.)

The FOBCCW macro instruction must be coded immediately following the DCs that define the buffer contents. This macro instruction creates a CCW string to print the buffer data when the operator specifies VER on the LOADBUF command. The format of the FOBCCW macro instruction is:

| | | |
|---|---|---|
| | FOBCCW | fobname [(print1,print2,...print12)] |

**where:**

fobname is the 1- to 4-character name assigned to the buffer by the FOB macro instruction.

[(print1,...print12)]
specifies the length of each line (up to 12 lines) printed to verify the buffer contents. The line length must be between 1 and 132 (the line length of a 3289 Model 4 printer). The default specification for verification is eight 64-byte lines of hexadecimal formatted data. The total number of hexadecimal bytes to be printed must not exceed 512. (There are two printed bytes for each of the 256 bytes of data in the buffer.)

Finally, insert the two macros just coded, FOB and FOBCCW, into the DMKPIA module. This module must be reloaded before the new buffer image can be used. DMKPIA is a pageable module with no executable code. DMKPIA must be on a page boundary and cannot exceed a full page in size. If DMKPIA exceeds a page boundary (4K), an error message is issued.

**Example:** The F64 Buffer

```
FOB     F64
DC      X'7F',63X'80'
DC      X'7F',9X'80',X'302E31322D3315'
DC      9X'80',X'342223353637210C',9X'80'
DC      X'1617393A3B',9X'80','3F3C0A0B2F303E'
DC      65X'80',X'2425262728292A2B2C'
DC      7X'80',X'18191A1B1C1D1E1F20'
DC      6X'80',X'38800D0E0F1011121314'
DC      6X'80',X'09000102030405060708'
DC      6X'80'
FOBCCW F64,(64,64,64,64,64,64,64,64)
```

## *UCC Buffer Images for the 3203 Printer*

The UCC buffer contains up to 240 characters and supports the 3203 printer. To add a new UCC buffer image, first code the UCC macro. This creates a 12-byte header for the buffer load that is used by CP. The format of the UCC macro is:

| | | |
|---|---|---|
| | UCC | uccname |

**where:**

uccname is a 1- to 4-character name that is assigned to the buffer load.

Next, supply the exact print image. The print image is supplied by coding DCs in hexadecimal or character format. The print image may consist of several DCs, the total length of the print image cannot exceed 240 characters.

The UCCCCW macro must immediately follow the 64-byte associative field, which must follow the print image. (See Note 3.) This macro creates a CCW string to print the buffer load image when VER is specified by the operator on the LOADBUF command. The format of the UCCCCW macro is:

| | | |
|---|---|---|
| | UCCCCW | uccname[,(print1,print2,...,print12)] |

**where:**

uccname is a 1- to 4-character name that is assigned to the buffer load by the UCC macro.

[(print1,...,print12)]
is the line length (or number of characters to be printed by the corresponding CCW) for the verify operation. Each count specified must be between 1 and 132 (the length of the print line on a 3203 printer) and the default line length is 48 characters. Up to 12 print fields may be specified. However, the total number of characters to be printed may not exceed 240.

Finally, insert the macros just coded, UCC and UCCCCW, into the DMKUCC module. This module must be reloaded. DMKUCC is a pageable module with no executable code. DMKUCC must be on a page boundary and cannot exceed a full page in size. If DMKUCC exceeds a page boundary (4K), an error message is issued.

### Examples of New UCC Buffer Images

**Example 1:** You do not have to specify the line length for verification of the buffer load. Insert the following code in DMKUCC:

```
UCC     EX01
DC      5CL'1234567890A...Z1234567890*/'
DC      X'00101010101010101010004000404000'  240-255
DC      X'40101010101010101010004040400000'  256-271
DC      X'40401010101010101010004000000000'  272-287
DC      X'10101010101010101010000000404000'  288-303
UCCCCW EX01
```

The buffer image is 5 representations of a 48-character string containing:

- The alphabetic characters
- The numeric digits, twice
- The special characters: * and /

Since the line length for the print verification is not specified on the UCCCCW macro, it defaults to 48 characters per line for 5 lines.

**Example 2:** Insert the following code in DMKUCC:

```
UCC     NUM1
DC      24CL'1234567890'
DC      X'00101010101010101010004000404000'  240-255
DC      X'40101010101010101010004040400000'  256-271
DC      X'40401010101010101010004000000000'  272-287
DC      X'10101010101010101010000000404000'  288-303
UCCCCW NUM1,(60,60,60,60)
```

The NUM1 print buffer consists of twenty-four 10-character entries. If, after DMKUCC is reloaded, the command

```
LOADBUF  00E  UCC  NUM1  VER
```

is specified, 4 lines of 60 characters (the 10-character string repeated 6 times) are printed to verify the buffer load.

**Note:** The UCS buffer for the 3203 MODEL 4 and MODEL 5 printers is programmed essentially the same as for the 1403 printer. You should follow the same procedures for programming this buffer, noting the differences listed below.

1. Instead of the UCS macro, code the UCC macro (which is equivalent to the UCS macro for the 1403).

2. The print train image should be 240 bytes long.

3. Immediately after the print image, the DUAL and UNCOMPARABLE TABLE (DUCT) should appear. The DUCT should be 64 bytes long and start at byte 240 in the UCS buffer.

4. Following the DUCT should be the UCCCCW macro (which is equivalent to the UCSCCW macro).

5. Finally, the completed macros and UCS data should be inserted into the DMKUCC module.

Note that when the UCCCCW macro is coded, you may specify that a maximum of 240 bytes is to be printed.

Also, the UCS buffer for the 3203 MODEL 5 printer must be 304 bytes long, while the UCS buffer for the 3203 MODEL 4 printer may be less (however, if the

LOADBUF command is issued, the buffer must be 304 bytes long). For information on the effects of the lower UCS buffer length, you should consult the *3203 Component Description and Operator's Guide*, GS33-1515.

For more information on coding the DUCT, consult the *3203 MODEL 5 Component Description and Operator's Guide*, (GA33-1529), or the *3203 Component Description and Operator's Guide*, (GA33-1515).

## *PIB Buffer Images for the 3262 Model I and II Printers*

The PIB buffer contains up to 132 characters and supports the 3262 printer. To add a new PIB buffer image, first code the PIB macro. This creates a 12-byte header for the buffer load that is used by CP. The format of the PIB macro is:

| | | |
|---|---|---|
| | PIB | pibname |

**where:**

pibname is a 1- to 4-byte character name that is assigned to the buffer load.

Next, supply the exact print image. The print image is supplied by coding DCs in hexadecimal or character format. The print image may consist of several DCs, the total length of the print image cannot exceed 288 characters.

The PIBCCW macro must immediately follow the print image. This macro creates a CCW string to print the buffer load image when VER is specified by the operator on the LOADBUF command. The format of the PIBCCW macro is:

| | | |
|---|---|---|
| | PIBCCW | pibname [,(print1,print2, ..., print12)] |

**where:**

pibname is a 1- to 4-character name that is assigned to the buffer load.

[,(print1,print2, ...,print12)]
specifies the length of each line (up to 12 lines) printer to verify the buffer contents. The line length must be between 1 and 132 (the length of the print line on a 3262 printer) and the default line length is 24 characters. Up to 12 print fields can be specified. However, the total number of characters to be printed may not exceed 288.

Finally, insert the macros just codes, PIB and PIBCCW, into the DMKPIB module. This module must be reloaded. DMKPIB is a pageable module with no executable code. DMKPIB must be on a page boundary and cannot exceed a full page in size. If DMKPIB exceeds a page boundary (4K), an error message is issued.

### Examples of New PIB Buffer Images

**Example 1:** You do not have to specify the line length for verification of the buffer load. Insert the following in DMKPIB:

```
PIB     EX01
DC      8CL36'1234567890...WXYZ'
PIBCCW  EX01
```

The buffer image is 8 representations of a 24-character string containing:

- The numeric digits
- The alphabetic characters

Since the line length for the print verification is not specified on the PIBCCW macro, it defaults to 24 characters per line for 8 lines.

**Example 2:** Insert the following code in DMKPIB:

```
PIB     EX02
DC      8CL36'1234567890...WXYZ'
PIBCCW EX02,(36,36,36,36,36,36,36,36)
```

The EX02 print buffer consists of eight 36-character entries. If, after DMKPIB is reloaded, the command:

```
LOADBUF   00E  PIB   EX02  VER
```

is specified, 8 lines of 36 characters are printed to verify the buffer load.

## Forms Control Buffer

It is possible to have a forms control buffer with both a virtual and real 3211-type (3203, 3211, 3262-1/5/11, 3289 Model 4, 4245) printer. A virtual 3211-type printer file can be printed on a real 1403; in fact, one way to provide forms control for a 1403 is to define it as a virtual 3211.

There is an FCB macro to support 3211-type forms control. The format of the FCB macro is:

| | | |
|---|---|---|
| | FCB | fcbname,space,length,(line,channel...),index |

**where:**

fcbname is the name of the forms control buffer. "fcbname" can be one to four alphameric characters.

space is the number of lines/inch. Valid specifications are 6 or 8. This operand may be omitted: the default is 6 lines/inch. When the space operand is omitted, a comma (,) must be coded. Spacing has no meaning for a virtual printer.

length is the number of print lines per page or carriage tape (1 to 255).

(line,channel...)
shows which print line (line) prints in each channel (channel...). "channel" values range from 1 to 12. Refer to the previous "Print Buffers and Forms Control" section for the VM/SP supplied buffer images for the FOB. The entries can be specified in any order.

index is an index value (from 1 to 31). "index" specifies the print position that is to be the first printed position. "index" is valid only for the 3211 printer. The "index" specification is not accepted for other printers. (The "index" specification can be overridden with the LOADBUF command).

VM/SP provides two real FCB images, FCB1 and FCB8. These FCBs are in pageable module DMKFCB. Installations may add additional FCB images to DMKFCB as long as the size of DMKFCB does not exceed the size of two pages.

A default virtual FCB image is provided for virtual 3211-type printers (3211, 3203, 3262-1/5/11, 3289 Model 4, and 4245). The image is used for the virtual printer if no FCB has been previously loaded for that virtual device. The image (66 bytes long with a channel 1 code at FCB position 1 with all other channels defined) is not stored in the spool file but only used for virtual processing of the print commands.

**Notes:**

1. The Forms Control Buffers *must* have compatibility of channel one; that is, channel one and line one must be the same physical line for all FCB's that are built, or forms misalignment results.

2. If the FCB macro is coded to have more than one channel designated for one print line, the macro includes only the last channel in the buffer for that print line. This is because a buffer byte can only be loaded with one channel code.

3. When an operator loads a default FCB, it is recommended that all channels be defined to prevent an undefined channel error.

**Example 1:** If you wanted your printer to print:

- 8 lines/inch
- 60 lines/page
- print line 3 in channel 1
- print line 60 in channel 9
- print line 40 in channel 12
- print position 10 the first print position

you would code the FCB macro (with a name, SPEC) as:

```
FCB SPEC,8,60,(3,1,40,12,60,9),10
```

If you want another forms control buffer, called LONG, to be exactly the same as SPEC (except that only 6 lines print per inch) you could code either of the following:

```
FCB LONG,6,60,(3,1,40,12,60,9),10
```

```
FCB LONG,,60,(3,1,40,12,60,9),10
```

**Example 2:** You could have your special forms control buffer (SPEC) loaded for either a virtual or real 3203, 3211, 3262, 3289 Model 4, or 4245 printer. The LOADVFCB command is for the virtual printer and the LOADBUF command is for the real printer.

The INDEX parameter is only valid for a 3211 printer. If INDEX is not specified for the 3211 printer, no indexing is done. If INDEX is specified without a value, the value coded in the FCB macro is used and if INDEX is specified with a value, the specified value overrides the value coded in the FCB macro.

If you specify INDEX for the virtual 3211 printer and again for the real 3211 printer, the output is indexed using the sum of the two specifications minus 1. For example, the command

```
LOADVFCB 00F FCB SPEC INDEX
```

indexes the virtual print file 10 positions because 10 was specified in the FCB macro for the SPEC forms control buffer. When this file is sent to the real printer, the operator issues the command

```
LOADBUF 00E FCB SPEC INDEX 20
```

which indexes the file an additional 20 positions. The value specified on the command line (20) overrides the value in the FCB macro (10). The output starts printing in print position 29 (10+20-1=29).

Because the 3203, 3262, 3289 Model 4, and 4245 printers do not have indexing capabilities, the LOADVFCB and LOADBUF commands with the INDEX option cause an invalid option error message from CP.

# IBM 3800 Printing Subsystem

The IBM 3800 Printing Subsystem is a high-speed, nonimpact printer that combines electro-photographic and laser technology. The 3800 printer can achieve speeds of up to 20,040 lines/minute, while several unique features give the user the ability to control the characteristics of printed output.

The features of the 3800 printer include:

- Forms control buffer - controls the amount of vertical space between printed lines. The user can specify vertical spacing of 6, 8, or 12 lines/inch.
- Multiple copy printing - allows the user to request multiple copies without the use of carbon paper. The 3800 uses its high speed to repeat-print the specified number of originals.
- Copy modification - allows the user to print or suppress predefined information on specified copies of a page. For example, a different name and address can be printed on each copy of a page.
- Forms overlay - allows the user to specify a form or grid to be printed (flashed) from a negative while output is being printed inside the form.
- Character arrangement tables - allow the user to specify which predefined character set to use for printing a data set. Each character set contains up to 64 printable characters.
- Character modification - allows character sets to be modified or extended to meet the user's needs.

For detailed information on the 3800 Printing Subsystem see:

- *Introducing the IBM 3800 Printing Subsystem and Its Programming*
- *Concepts of the IBM 3800 Printing Subsystem*
- *IBM 3800 Printing Subsystem Programmer's Guide, OS/VS1, OS/VS2*
- *Reference Manual for the IBM 3800 Printing Subsystem.*

VM/SP supports the 3800 printer as a dedicated device, as a real spooling device, and as a virtual spooling device.

## Using the 3800 Printer as a Dedicated Device

VM/SP allows a virtual machine that is configured to support a real 3800 to attach the 3800 for that machine's exclusive use. When used as a dedicated device, VM/SP supports all of the facilities of the 3800.

## Using the 3800 Printer as a Real Spooling Device

VM/SP allows users of spool files to print their files on an IBM 3800 Printing Subsystem. The copy modification, forms overlay, character modification, and multiple copy features are fully supported. However, when the 3800 is used as a real spool-

ing device, only one character arrangement table may be specified for each spool file. In addition, the entire spool file must be printed with the same line spacing on each page.

Parameters on five commands enable the user to take advantage of the 3800 printer's capabilities. The CHANGE, SPOOL, and START commands allow the user to specify the character arrangement table, copy modifications, forms overlay, and FCB to be used for printing. The BACKSPAC and QUERY commands also support the 3800 printer.

Three utilities, GENIMAGE, IMAGELIB, and IMAGEMOD, construct and modify the character arrangement tables, graphic modifications, copy modifications, and FCBs used by the 3800. DIAGNOSE Code X'74' is invoked by IMAGELIB and IMAGEMOD to load and save this control information as a named system.

Finally, the NAME3800 macro instruction allows the user to create the named system that contains the control information needed to print a spool file.

### *Specifying Printer Options*

Five parameters on the SPOOL and CHANGE commands support the 3800 printer as a real spooling device. (See the *CP Command Reference for General Users* for detailed coding information.)

- FLASH - identifies the form overlay, if any, to be used when printing the file
- CHARS - names the character arrangement table to be used to print the file
- MODIFY - indicates the copy modification module, if any, to be used when printing the file
- FCB - specifies the name of the forms control buffer to be used for the file
- COPY - indicates the number of copies to be printed.

The START command includes parameters that enable the VM/SP operator to name the character arrangement table and the FCB to be used for the separator page. The operator can also identify, via the FLASH operand of the START command, the forms overlay currently loaded in the 3800. In addition, the operator uses the IMAGE parameter to specify the named system to be used to print the spool file. Finally, by specifying the PURGE parameter, the operator can purge all spool files that cause errors when loaded into the 3800. See the *VM/SP Operator's Guide* for further information on the START command.

### *Creating Control Tables*

VM/SP uses the OS/VS utility IEBIMAGE to create and dynamically modify character arrangement tables, copy modifications, graphic modifications, and FCBs. Three service programs, GENIMAGE, IMAGELIB, and IMAGEMOD, interface with IEBIMAGE.

GENIMAGE creates or modifies text files on a CMS disk. These text files contain the images to be used by the 3800 Printing Subsystem.

IMAGELIB loads the new or changed text files created by GENIMAGE into virtual storage. When all the files are loaded, DIAGNOSE Code X'74' is invoked to save these files as a named system.

IMAGEMOD makes selective modifications to an existing 3800 named system without completely regenerating the named system. While IMAGELIB creates a new named system each time it is invoked, IMAGEMOD allows users to add, delete, replace, and list members of a 3800 named system. IMAGEMOD uses DIAGNOSE X'74' to load and save the named system.

See the *VM/SP Planning Guide and Reference* for more information on IMAGELIB, GENIMAGE and IMAGEMOD.

### *Storing and Loading Control Tables*

As part of VM/SP support of the 3800 printer, character arrangement tables, copy modifications, graphic modifications, and FCBs are stored in a named system.

Prior to printing a spool file, the VM/SP operator specifies a named system on the IMAGE parameter of the START command. The control tables specified for the file (via the SPOOL and CHANGE commands) are loaded into the 3800 from that named system and the file is printed.

The NAME3800 macro instruction establishes the named system at system generation. See the *VM/SP Planning Guide and Reference* for further information.

### *Recovering from I/O Errors*

Because the actual printing of lines on the page is slower than the output of lines from the processor, spool files are placed into a delayed purge queue to await printing by the 3800. Only when the maximum number of files are in the queue will the first one actually be purged. The size of the queue can be specified at system generation time via the DPMSIZE parameter on the RDEVICE macro instruction. DPMSIZE can have a maximum value of nine.

Because spool files are queued, the BACKSPAC command may be used for the 3800 printer to restore pages that are lost when an I/O error occurs. In addition, the operator may specify the EOF parameter, which indicates that backspacing should begin at the end of the file and continue for the number of pages specified. See the *VM/SP Operator's Guide* for more information on the BACKSPAC command.

### *Displaying Printer Control Information*

The QUERY command enables G-, B-, and D-privilege users to display the names of the character arrangement table, copy modification, and FCB currently in effect for a spool file or a virtual printer. In addition, the VM/SP operator can use the QUERY command to determine the image library used and the forms loaded on a real 3800.

See the *VM/SP Operator's Guide* for details on the QUERY command.

## Using the 3800 Printer as a Virtual Spooling Device

VM/SP enables a user to create printer spool files on a virtual 3800 printer defined for his virtual machine. VM/SP provides full support for the copy modifications, forms overlay, character modifications, and multiple copy features. In addition,

when the 3800 is defined as a virtual spooling device, the user can specify that up to four different Writable Character Generation Modules (WCGM's) are available for defining characters to print on a line or page. Each WCGM holds 64 characters. Finally, the user can vary the vertical spacing of lines on a page.

VM/SP makes these 3800 features available both to CMS users and to virtual machines running an operating system with full 3800 support. When a virtual machine running an operating system with full 3800 support issues commands that generate 3800 load commands, the virtual machine operating system passes these load commands and their associated data to CP. CP includes these load commands in the virtual spool file.

Four commands and macros make it possible for CMS users to load a virtual 3800 printer and to print virtual 3800 spool files. The SETPRT command and parameters on the PRINTL macro, and PRINT and FILEDEF commands make it possible to use the features of a 3800 printer.

Parameters on the CP START command allow the VM/SP operator to control the printing of virtual 3800 spool files. The QUERY command enables a user to determine the characteristics of his virtual 3800 printer or spool file.

### *Defining a Virtual 3800 Printer*

To use the features of the 3800 printer as a virtual spooling device, the installation or user must define a virtual 3800 for the user's virtual machine. The class G DEFINE command enables a user to specify virtual 3800 characteristics, including how many WCGMs the virtual printer has, and whether or not CP will reflect all data checks to the virtual machine. (See the *VM/SP CP Command Reference for General Users* for details on the DEFINE command.)

The SPOOL control statement of the DIRECT command allows an installation to define a virtual 3800 in a user's directory. (See the *VM/SP Planning Guide and Reference* for details on the SPOOL control statement.)

### *Loading the Virtual 3800 and Printing Virtual 3800 Spool Files*

The CMS SETPRT command allows a CMS user to include 3800 control information at the beginning of a spool file. When the file prints on a real 3800, the real printer uses this control information to determine which character sets, copy modifications, and FCB to load and which copies to print with the forms overlay. The SETPRT command also allows a user to specify copy groups and the number of copies to print. (See the *VM/SP CMS Command and Macro Reference* for details on the SETPRT command.)

Once the SETPRT command has been issued, CMS allows a user to select the character set used to print each line of a virtual 3800 spool file. The TRC option of the CMS PRINT command and the PRINTL macro indicates that each line in the file has a TRC (Table Reference Character) as the first byte of data. The TRC bytes correspond to the order in which character sets have been loaded by the SETPRT command. When the file prints on the real 3800 printer, the value of the TRC byte determines which character set the 3800 selects to print each line.

For CMS users running OS programs, coding the OPTCD J parameter of the FILEDEF command indicates to the QSAM PUT macro and the BSAM WRITE macro that each output line contains a TRC byte.

Note that files created on a virtual 3800 can print on any real spooling device supported by VM/SP. However, if a file that was created on a virtual 3800 is printed on another real printer (for example, a 1403 or a 3211) all 3800 unique control information is ignored by the real printer. Further, while print lines of 204 bytes are allowed by the 3800, lines will be truncated if printed on a real printer with a smaller maximum line length (for example, a 1403 or a 3211).

### *Recovering from I/O Errors*

When an I/O error occurs while a virtual 3800 spool file is printing on a real 3800, the VM/SP operator can specify the BACKSPAC command with the FILE parameter to recover from the error. The BACKSPAC command with the FILE parameter restarts the spool file at the beginning and reloads the real 3800 with the control information originally specified in the SETPRT command.

### *Displaying Control Information*

The CP QUERY command enables G-, B-, and D-privilege users to display the names of the character arrangement tables, copy modifications, and FCB currently in effect for a spool file or virtual printer. In addition, the QUERY command displays the characteristics of the virtual 3800 that were established when the virtual device was defined. Finally, the user can determine where 3800 load commands are found in a file (not at all, at the beginning, or throughout).

The VM/SP operator can use the QUERY command to determine the image library and form loaded on the real 3800. Further, the VM/SP operator can determine which virtual 3800 spool files are being accepted for printing by the real printer: those containing no 3800 load commands, those with 3800 load commands only at the beginning, or those with 3800 load commands throughout (as long as the class and FORMS match). The VM/SP operator specifies on the CP START command which spool files the real 3800 accepts for printing.

## Journaling Logon, Autolog, and Link Commands

LOGON, AUTOLOG, and LINK Journaling attempts to detect and record certain occurrences of the LOGON, AUTOLOG, or LINK commands. Using the recorded information, an installation may be able to identify attempts to logon to VM/SP by users that issue invalid passwords. Also, the installation may be able to identify users that successfully issue the LINK command to protected minidisks not owned by that user.

Briefly, LOGON, AUTOLOG, and LINK journaling works like this. While journaling is turned on, CP monitors all occurrences of the LOGON, AUTOLOG, and LINK commands. CP keeps count of the number of times a user issues one of these commands with an invalid password. When this count exceeds an installation defined threshold value, CP optionally:

- Writes a record to the accounting data set to record the incident
- Rejects subsequent LOGON, AUTOLOG, or LINK commands issued by the user
- Sends a message to an installation-defined user identification to alert the installation to the incident

Also, each time CP detects that a user has successfully issued a LINK command to a protected minidisk not owned by that user, CP optionally records the incident by writing a record to the accounting data set. A protected minidisk is a minidisk whose password is anything but ALL for the type of LINK attempted.

For a description of the accounting records that CP writes for LOGON, AUTOLOG, and LINK journaling, see the section "Accounting Records."

The SYSJRL macro instruction, the SET command, and the QUERY command enable an installation to control LOGON, AUTOLOG, and LINK journaling. To make journaling available and to specify options, code the SYSJRL macro instruction in module DMKSYS. Instructions for coding this macro instruction are in the *VM/SP Planning Guide and Reference.* To turn journaling on or off, use the class A SET command. To determine whether journaling is on or off, use the class A QUERY command.

## Suppressing Passwords Entered on the Command-Line

CP optionally rejects LOGON or LINK commands that have the password entered on the same line as the command. Rejecting these commands prevents passwords from being displayed or from being printed without masking -- masking a password means overprinting the password so it cannot be read.

This capability is also available to virtual machines that issue LINK commands via DIAGNOSE Code X'08'. For a description of DIAGNOSE Code X'08', see the section "DIAGNOSE Instruction in a Virtual Machine."

To request password suppression, specify it as an option on the SYSJRL macro instruction in module DMKSYS during system generation of VM/SP. Once requested, password suppression is always on; an operator cannot turn it off. Refer to the *VM/SP Planning Guide and Reference* for information on how to use and code SYSJRL in DMKSYS.

# Part 2. Conversational Monitor System (CMS)

Part 2 contains the following information:

- Introduction to CMS
- Interrupt Handling
- Functional Information (How CMS Works)
  - Register usage
  - DMSNUC structure
  - Storage structure
  - Free storage management
  - SVC handling
- CMS IUCV Support
- Using the DASD Block I/O System Service from CMS
- OS Macro Simulation
- VSE Support Under CMS
- CMS Support for OS and DOS VSAM Functions
- Saving the CMS system
- Batch Monitor
- The Programmable Operator Facility
- Auxiliary Directories
- Assembler Virtual Storage Requirements

# Introduction To CMS

The Conversational Monitor System (CMS), the major subsystem of VM/SP, provides a comprehensive set of conversational facilities to the user. Several copies of CMS may run under CP, thus providing several users with their own time sharing system. CMS is designed specifically for the VM/SP virtual machine environment.

Each copy of CMS supports a single user. This means that the storage area contains only the data pertaining to that user. Likewise, each CMS user has his own machine configuration and his own files. Debugging is simpler because the files and storage area are protected from other users.

Programs can be debugged from the terminal. The terminal is used as a printer to examine limited amounts of data. After examining program data, the terminal user can enter commands on the terminal that will alter the program. This is the most common method used to debug programs that run in CMS.

CMS, operating with the VM/SP Control Program, is a time sharing system suitable for problem solving, program development, and general work. It includes several programming language processors, file manipulation commands, utilities, and debugging aids. Additionally, CMS provides facilities to simplify the operation of other operating systems in a virtual machine environment when controlled from a remote terminal. For example, CMS capabilities are used to create and modify job streams, and to analyze virtual printer output.

Part of the CMS environment is related to the virtual machine environment created by CP. Each user is completely isolated from the activities of all other users, and each machine in which CMS executes has virtual storage available to it and managed for it. The CP commands are recognized by CMS. For example, the commands allow messages to be sent to the operator or to other users, and virtual devices to be dynamically detached from the virtual machine configuration.

## The CMS Command Language

The CMS command language offers terminal users a wide range of functions. It supports a variety of programming languages, service functions, file manipulation, program execution control, and general system control. The CMS commands that are useful in debugging are discussed in the "Debugging with CMS" section of "Part 3. Debugging with VM/SP". For detailed information on all other CMS commands, refer to the *VM/SP CMS Command and Macro Reference.*

Figure 35 describes CMS command processing.

## The File System

The Conversational Monitor System interfaces with virtual disks, tapes, and unit record equipment. The CMS residence device is kept as a read-only, shared, system disk. Permanent user files may be accessed from up to 25 active disks. Logical access to those virtual disks is controlled by CMS, while CP facilities manage the device sharing and virtual-to-real mapping.

User files in CMS are identified with three designators. The first is filename. The second is a filetype designator that may imply specific file characteristics to the CMS file management routines. The third is a filemode designator that describes the location and access mode of the file.

User files can be created directly from the terminal with the System Product Editor (XEDIT). XEDIT provides extensive context editing services. File characteristics such as record length and format, tab locations, and serialization options can be specified. The system includes standard definitions for certain filetypes. The size of user files is determined by the BLKSIZE. When a BLKSIZE of 800 bytes is specified, a single user file is limited to a maximum of 65533 records and must reside on one virtual disk. The file management system limits the number of files on the virtual disk to 3400. When a BLKSIZE of 1024, 2048, or 4096 bytes is specified, a single user file is limited to a maximum of $2^{31}$-1 CMS records and must reside on one virtual disk. The maximum number of data blocks available in a variable format file on a 512-byte blocksize minidisk is about 15 times less than $2^{31}$-1. The file management system does not limit the number of files on the disk. The number of files on a disk is limited by the capacity of the disk.

The compilers available under CMS default to particular input filetypes, such as ASSEMBLE, but the file manipulation and listing commands do not. Files of a particular filetype form a logical data library for a user; for example, the collection of all COBOL source files, or of all object (TEXT) decks, or of all EXEC procedures. This allows selective handling of specific groups of files with minimum input by the user.

CMS automatically allocates compiler work files at the beginning of command execution on whichever active disk has the greatest amount of available space, and deallocates them at completion. Compiler object decks and listing files are normally allocated on the same disk as the input source file or on the primary read/write disk, and are identified by combining the input filename with the filetypes TEXT and LISTING. These disk locations may be overridden by the user.

Virtual disks may be shared by CMS users; the facility is provided by VM/SP to all virtual machines, although a user interface is directly available in CMS commands. Specific files may be spooled between virtual machines to accomplish file transfer between users. Commands allow such file manipulations as writing from an entire disk or from a specific disk file to a tape, printer, punch, or the terminal. Other commands write from a tape or virtual card reader to disk, rename files, copy files, and erase files. Special macro libraries and text or program libraries are provided by CMS, and special commands are provided to update and use them. CMS files can be written onto and restored from unlabeled tapes via CMS commands.

*Caution*: Multiple write access under CMS can produce unpredictable results.

Problem programs that execute in CMS can create files on unlabeled tape in any record and block size; the record format can be fixed, variable, or undefined.

## Migration from the 800-byte File System to the Extended File System

This section discusses the points to consider when migrating to the VM/SP file system from VM/370 Release 6 or earlier versions of CMS. Note that the VM/SP file system is directly compatible with the VM/370 System Extensions and the Basic System Extensions, Release 2 file system. The VM/SP file system provides greater file capacities, four disk blocksizes for minidisks, and improved performance over most earlier versions of CMS.

The VM/SP file system contains support for the traditional CMS disk format and for an enhanced format of CMS disk. It is with this enhanced or extended format that additional functions and capacities are available. The VM/SP extended file

format has a completely revised internal file structure. The extended format structure is highly compatible with previous versions of CMS and retains the strengths of earlier CMS file systems.

The VM/SP file system provides the following for extended format disks:

- The logical disk capacity has been increased. There is no effective limit to the size of a CMS minidisk except for the size of the physical device.
- The logical file capacity has been increased. There is no effective limit on the size of a file except for the amount of space on the minidisk. (The record number must be less than $2^{31}-1$.)

  Performance when randomly accessing variable length records has been improved.
- The number of minidisks that can be accessed at any one time by a single user has been increased from 10 to 26.
- In VM/370, the number of files per minidisk was limited to 3400. With the VM/SP file system extensions, there is no effective limit except for the constraints of storage and disk space.
- In VM/370, the minimum number of physical disk blocks needed to hold very small files was two; it is now one. Note that this may or may not cause a saving of disk space depending on the physical blocksize chosen and the exact size of the small file.
- Internal algorithms controlling the updating of blocks on disk when files are closed have been redesigned. The redesign provides a significant increase in performance for most CMS users.
- The physical blocksize of the VM/370 file system was 800. The VM/SP file system supports four additional sizes:
  - 512 bytes
  - 1024 bytes
  - 2048 bytes
  - 4096 bytes

## *Migration Considerations*

### Disk Formats

The in-storage control structures of the VM/SP file system are considerably different than those of the VM/370 800-byte file system. However, because backward compatibility has been maintained in the VM/SP file system, disks that are formatted under earlier versions of CMS can be used with VM/SP. Disks that are formatted with a blocksize of 800-bytes under VM/SP can be used with earlier versions of CMS. Note, however, that the default blocksize for VM/SP is 1024 bytes when minidisks are formatted using the VM/SP FORMAT command.

Although the internal blocksize is transparent to most users and programs, installation utilities that dump and restore disks may depend upon the physical disk blocksize and the internal disk control block structure.

### MACLIB and TXTLIB Files

The internal format of MACLIB and TXTLIB files has been augmented by the addition of a new format to allow larger libraries. Earlier CMS library formats are supported by VM/SP whether they exist on 800-byte or extended format disks. Updating an old format MACLIB or TXTLIB on an extended disk does not change the internal format. The creation of a library on an extended format disk causes the construction of a library with the new format.

Under VM/SP, new format libraries are supported on 800-byte format disks. This condition can only occur if a new format library is copied (via COPYFILE, MOVEFILE, and so on) from an extended format disk to an 800-byte format disk. Note, however, that the new format libraries are not supported by earlier versions of CMS even if on 800-byte disks.

### TAPE Command

The format of tapes created by the TAPE command has been augmented by the addition of a larger blocksize of 4096-bytes. Tapes created by earlier versions of CMS are properly read onto any disk format by VM/SP. Tapes created by VM/SP are *not* readable by earlier versions of CMS *unless* they are dumped with blocksize of 800. The default blocksize is 4096 when minidisks are dumped using the VM/SP TAPE command.

### DISK Command

The format of spool files created by the DISK command differs slightly from earlier versions of CMS. However, files dumped by previous versions of CMS are properly read by VM/SP and files dumped by VM/SP are properly read by earlier versions of CMS *provided* that the file meets the size constraints of the 800-byte disk (especially, the dumped file must not be greater than 65,533 records).

### Program I/O

Programs that do I/O to CMS disks fall into three categories:

- Those that do CMS I/O (for example, FSOPEN, FSREAD, FSWRITE)
- Those that do OS I/O (for example, OPEN DCB, READ, GET)
- Those that do VSE I/O (for example, OPEN DTF, READ, GET)

Only programs that do CMS disk I/O directly have any compatibility or migration considerations. Programs that issue OS or VSE I/O calls can immediately take advantage of the capacity of the extended file system as soon as the files are put on a VM/SP extended format disk.

Programs that issue CMS I/O macros or calls continue to work on both the 800-byte and the VM/SP extended file systems but are not able to take advantage of all of the VM/SP file capabilities without conversion. This includes the use of the FSSTATE macro, which returns the correct format File Status Table (FST) whatever the disk format.

However, it should be noted that, in general, programs that deal with internal system control blocks, (such as File Status Table (FST) blocks, Active Disk Table (ADT) blocks, or Active File Table (AFT) blocks) should *not* be used under VM/SP without careful examination of the program, and conversion or elimination of program references to the internal blocks.

See the *VM/SP CMS Command and Macro Reference* for information on how to use file system macros with the extended file system. See the *VM/SP System Logic and Problem Determination Guide, Volume 2,* to learn about the structure of the VM/SP file system.

**Programs That Reference System Information**

Existing programs that reference internal CMS file system control information will probably not function correctly under VM/SP.

*Note*: Unconverted programs that run with the SYSTEM attribute (privileged) or do direct I/O (DIAGNOSE) can destroy data on (or the logical structure of) a minidisk, making part or all of the data on that minidisk inaccessible.

Before running them on VM/SP examine any programs that perform functions similar to the TAPE and DISK commands, programs that copy files, programs that copy minidisks, and programs that are used to dump and restore files for backup.

Areas to be examined include:

- The format of the File Status Table (FST) has been changed in several ways including its length. Programs that reference FST copies returned by the STATE command or FSSTATE/FSOPEN macros continue to function because new format FSTs are converted to old formats in the copy returned to the user.

  Programs that reference fields in real FSTs (those in the FST hyperblocks or AFTs) may not function properly. Programs that change fields in FSTs can destroy the integrity of the file system.

  Note that careful evaluation of both the program and CMS file system internal processes may be necessary to determine what must be done with such programs.

- The format and contents of the Active File Table (AFT) control block have been significantly changed for extended format disks. The order of fields in the AFT for 800-byte block disks has been changed. Programs referencing the AFT should be carefully examined and must at least be reassembled before running them even with 800-byte format disks.

- The format of the Active Disk Table (ADT) has been significantly altered. Many fields with new meaning have been added, and many existing fields have been relocated. One such field is the disk volume label VOLID, which has moved.

- The format of most other VM/370 file system control information is different from the VM/SP format file system. Programs that reference such data should be carefully examined and altered before running them.

- Programs that install auxiliary directories by changing the SSTAT field in NUCON will not function properly if the S-disk is an extended format disk. The CMS routine DMSLADAD should be used to install *all* auxiliary directories.

- All programs that reference VM/370 CMS control block macros should be reassembled under VM/SP. If the only control block referenced is NUCON,

the assembly is not necessary. The DMSSP and CMSLIB MACLIBs should be specified as the macro libraries to be searched for CMS macro references with the CMS GLOBAL command. The DMSSP MACLIB should precede the CMSLIB MACLIB, followed by any other MACLIBs needed for the assembly.

**Auxiliary Directories (AUXDIRTS)**

Auxiliary directories are logical extensions of file system directories that reside as a part of certain programs. User programs containing auxiliary directories continue to function on extended format disks provided that the module containing the auxiliary directory is regenerated in the manner normal for any movement of such a module. CMS correctly converts internal formats so that auxiliary directories function properly as long as they are installed by calling the CMS routine DMSLADAD.

**LISTFILE Command**

The LISTFILE command is compatible with previous versions except that the columns in which information is placed have been moved. Programs that use the CMS EXEC file produced by LISTFILE should be examined, especially, those that sort CMS EXEC files by data or file size.

**QUERY DISK Command**

The QUERY DISK command has been completely changed. See the QUERY command in the *VM/SP CMS Command and Macro Reference* manual for more information.

**Other Command Changes**

Several other commands have been changed. Programs that examine spooled console output for the typed results of certain commands might require changes.

## *Coexistence of VM/SP CMS and Earlier Versions of CMS*

During conversion from an earlier version of CMS to VM/SP it might be desirable, depending on local conditions, to run both versions of CMS for a period of time. However, it is important to remember that *system* modules from earlier systems should *never* be executed on the VM/SP system and vice-versa. Such modules are incompatible and will damage system and/or user data if run in the incorrect environment. Among others, consider the following points to allow easier switching back and forth between versions:

- All disks should be formatted with earlier versions of CMS or by specifying a blocksize of 800. No extended format disks should be used.
- All use of the TAPE command should be from the earlier version of CMS or should specify a blocksize of 800.
- Programs or EXEC files that reference CMS EXEC files, or programs that reference or change system control blocks, require special handling. One of the following actions should be taken:
  - Segregate such programs or EXEC files onto separate disks (one per CMS version) and access the one that corresponds to the CMS version you are currently using, or

- Make the programs or EXEC files aware of the difference in format so that they can properly execute under either system.

### *Converting CMS Files*

Although VM/SP can be run with only the 800-byte file system, as previously discussed, CMS disk formats and files must be converted to take advantage of the performance and capacity enhancements of VM/SP.

#### Converting Disk Formats

The conversion of disk formats can be achieved in several ways. The two main ones are:

- Allocate a second minidisk, format it under VM/SP using the FORMAT command with the desired blocksize, and use the COPY command to copy files from the old format disk to the extended format disk. The old format disk can then be deallocated.

- Dump the files from the old format disk to tape using the TAPE DUMP command. Format the disk under VM/SP with the desired blocksize. Load the files from the tape using the TAPE LOAD command.

#### Converting MACLIB and TXTLIB Files

MACLIB and TXTLIB files must be re-created to get them into the new library formats. Under VM/SP, use the VMFMAC EXEC procedure as described in the *VM/SP Installation Guide*. However, there is no need to do so unless the library needs the expanded capacity provided by the new format.

#### Program Conversions

Programs that do CMS I/O do not need to be converted (except, as previously discussed, for those referencing internal control blocks) to be run against files on extended format disks. All performance advantages are achieved by merely moving the files to an extended format disk. In addition, the maximum size of the file is limited only by the 65,533 record limit and not by the old 16,060 block limit.

Existing programs that need to access files larger than 65,533 records must be converted to take advantage of the greater capabilities of the VM/SP file system. See the *VM/SP CMS User's Guide* for more information.

#### Auxiliary Directories

Programs that use auxiliary directories must be regenerated when moved to an extended format disk. This regeneration would be required anyway because of movement from one disk to another.

## Program Development

The Conversational Monitor System includes commands to create and compile source programs, to modify and correct source programs, to build test files, to execute test programs and to debug from the terminal. The commands of CMS are especially useful for OS and VSE program development, but also may be used in combination with other operating systems to provide a virtual machine program development tool.

CMS uses the OS and VSE compilers via interface modules; the compilers themselves normally are not changed. In order to provide suitable interfaces, CMS

includes a certain degree of OS and VSE simulation. For OS, the sequential, direct, and partitioned access methods are logically simulated; the data records are physically kept in the chained fixed-length blocks, and are processed internally to simulate OS data set characteristics. For VSE, the sequential access method is supported. CMS supports VSAM catalogs, data spaces, and files on OS and DOS disks using the Access Method Services portion of the VSE/VSAM program product. OS Supervisor Call functions such as GETMAIN/FREEMAIN and TIME are simulated. The simulation restrictions concerning what types of OS object programs can be executed under CMS are primarily related to the OS/PCP, MFT, and MVT Indexed Sequential Access Method (ISAM) and the telecommunications access methods, while functions related to multitasking in OS and VSE are ignored by CMS. For more information, see "OS Macro Simulation under CMS" and "VSE Support under CMS".

## ABEND Processing

When CMS abnormally terminates, the following steps are taken:

1. After checking for any SPIE, STXIT PC, STAE, or STXIT AB exits that apply, CMS calls DMSABN, the abend recovery routine.

2. Before typing out any abend message at the terminal, DMSABN checks for any ABEND exit routines, set via the ABNEXIT macro.

3. If a list of exit routines exists, the current ABEND exit routine (that is, the last one set) gains control. If no ABEND exit routines exist, CMS abend recovery occurs.

### *ABEND Exit Routine Processing*

An ABEND exit routine may be established to intercept abends before CMS abend recovery begins. An ABEND exit routine receives control with the nucleus protect key and is disabled for interrupts. Information about the abend is available to the exit routine in the DMSABW CSECT in DMSNUC. The address of this area is passed to the exit routine via register 1.

An ABEND exit routine may choose to avoid CMS ABEND recovery and continue processing normally. To do this, the exit must issue the ABNEXIT RESET macro. This tells CMS to clear the ABEND condition. The exit routine may also return to CMS to continue ABEND processing.

If the exit routine returns to CMS and another ABEND exit routine exists, it is given control next. Each exit on the list is given control in sequence until all the exits have been given control or until an exit chooses to avoid CMS ABEND recovery, by issuing ABNEXIT RESET, and continues processing.

If a program check occurs in an exit routine, and ABNEXIT RESET was not issued in this exit routine, DMSABN gives control to the next exit routine on the list. If no other exit routine exists, CMS abend recovery occurs.

You cannot set or clear ABEND exit routines in an ABEND exit routine. You can reset an ABEND exit routine only in an exit routine.

If no ABEND exit routine exists, or if the ABEND exit routine returns to CMS to continue ABEND processing, DMSABN types out the abend message followed by the line:

```
CMS
```

This line indicates to the user that the next command can be entered.

Now, there are two options available:

- Type in the DEBUG command. DMSABN passes control to DMSDBG to make the facilities of DEBUG available. DEBUG's PSW and registers are as they were at the time the recovery routine was invoked. In DEBUG mode, you may alter the PSW or registers. Then, type GO to continue processing, or type RETURN to return to DMSABN. DMSABN continues the abend recovery.

- Type in any command (other than DEBUG). DMSABN performs its abend recovery function, and then passes control to DMSINT to execute the command that was typed in.

# Interrupt Handling In CMS

CMS receives virtual SVC, input/output, program, machine, and external interruptions and passes control to the appropriate handling program.

## SVC Interruptions

The Conversational Monitor System is SVC (supervisor call) driven. SVC interruptions are handled by the DMSITS resident routines. Two types of SVCs are processed by DMSITS: internal linkage SVC 202 and 203, and any other SVCs. The internal linkage SVC is issued by the command and function programs of the system when they require the services of other CMS programs. (Commands entered by the user from the terminal are converted to the internal linkage SVC by DMSINT). The OS SVCs are issued by the processing programs (for example, the Assembler).

### *Internal Linkage SVCs*

When DMSITS receives control as a result of an internal linkage SVC (202 or 203), it saves the contents of the general registers, floating-point registers, and the SVC old PSW, establishes the normal and error return addresses, and passes control to the specified routine. (The routine is specified by the first 8 bytes of the parameter list whose address is passed in register 1 for SVC 202, or by a halfword code following SVC 203.)

For SVC 202, if the called program is not found in the internal function table of nucleus (resident) routines, then DMSITS attempts to call in a module (a CMS file with filetype MODULE) of this name via the LOADMOD command.

If the program was not found in the function table, nor was a module successfully loaded, DMSITS returns an error code to the caller.

To return from the called program, DMSITS restores the calling program's registers, and makes the appropriate normal or error return as defined by the calling program.

### Other SVCs

The general approach taken by DMSITS to process other SVCs supported under CMS is essentially the same as that taken for the internal linkage SVCs. However, rather than passing control to a command or function program, as is the case with the internal linkage SVC, DMSITS passes control to the appropriate routine. The SVC number determines the appropriate routine.

In handling non-CMS SVC calls, DMSITS refers first to a user-defined SVC table (if one has been set up by the DMSHDS program). If the user-defined SVC table is present, any SVC number (other than 202 or 203) is looked for in that table. If it is found, control is transferred to the routine at the specified address.

If the SVC number is not found in the user-defined SVC table (or if the table is nonexistent), DMSITS either transfers control to the CMSDOS shared segment (if SET DOS ON has been issued), or the standard system table (contained in DMSSVT) of OS calls is searched for that SVC number. If the SVC number is found, control is transferred to the corresponding address in the usual manner. If the SVC is not in either table, then the supervisor call is treated as an abend call.

The DMSHDS initialization program sets up the user-defined SVC table. It is possible for a user to provide his own SVC routines.

## Input/Output Interruptions

All input/output interruptions are received by the I/O interrupt handler, DMSITI. DMSITI saves the I/O old PSW and the CSW (channel status word). It then determines the status and requirements of the device causing the interruption and passes control to the routine that processes interruptions from that device. DMSITI scans the entries in the device table until it finds the one containing the device address that is the same as that of the interrupting device. The device table (DEVTAB) contains an entry for each device in the system. Each entry for a particular device contains, among other things, the address of the program that processes interruptions from that device.

When the appropriate interrupt handling routine completes its processing, it returns control to DMSITI. At this point, DMSITI tests the wait bit in the saved I/O old PSW. If this bit is off, the interruption was probably caused by a terminal (asynchronous) I/O operation. DMSITI then returns control to the interrupted program by loading the I/O old PSW.

If the wait bit is on, the interruption was probably caused by a nonterminal (synchronous) I/O operation. The program that initiated the operation most likely called the DMSIOW function routine to wait for a particular type of interruption (usually a device end). In this case, DMSITI checks the pseudo-wait bit in the device table entry for the interrupting device. If this bit is off, the system is waiting for some event other than the interruption from the interrupting device; DMSITI returns to the wait state by loading the saved I/O old PSW. (This PSW has the wait bit on.)

If the pseudo-wait bit is on, the system is waiting for an interruption from that particular device. If this interruption is not the one being waited for, DMSITI loads the saved I/O old PSW. This again places the machine in the wait state. Thus, the program that is waiting for a particular interruption is kept waiting until that interruption occurs.

If the interruption is the one being waited for, DMSITI resets both the pseudo-wait bit in the device table entry and the wait bit in the I/O old PSW. It then loads that PSW. This causes control to be returned to the DMSIOW function routine, which, in turn, returns control to the program that called it to wait for the interruption.

## Terminal Interruptions

Terminal input/output interruptions are handled by the DMSCIT module. All interruptions other than those containing device end, channel end, attention, or unit exception status are ignored. If device end status is present with attention and a write CCW was terminated, its buffer is unstacked. An attention interrupt causes a read to be issued to the terminal, unless attention exits have been queued via the STAX macro. The attention exit with the highest priority is given control at each attention until the queue is exhausted, then a read is issued. Device end status indicates that the last I/O operation has been completed. If the last I/O operation was a write, the line is deleted from the output buffer and the next write, if any, is started. If the last I/O operation was a normal read, the buffer is put on the finished read list and the next operation is started. If the read is caused by an attention interrupt, the line is first checked to see if it is an immediate command (user-defined or built-in). If it is a user-defined immediate command, control is

passed to a user specified exit, if one exists. Upon completion, the exit returns to DMSCIT. If it is a built-in immediate command (HX, for example), appropriate processing is performed by DMSCIT. Unit exception indicates a canceled read. The read is reissued, unless it had been issued with ATTREST=NO, in which case unit exception is treated as device end.

## Reader/Punch/Printer Interruptions

Interruptions from these devices are handled by the routines that actually issue the corresponding I/O operations. When an interruption from any of these devices occurs, control passes to DMSITI. Then DMSITI passes control to DMSIOW, which returns control to the routine that issued the I/O operation. This routine can then analyze the cause of the interruption.

## User-Controlled Device Interruptions

Interrupts from devices under user control are serviced the same as CMS devices except that DMSIOW and DMSITI manipulate a user-created device table, and DMSITI passes control to any user-written interrupt processing routine that is specified in the user device table. Otherwise, the processing program regains control directly.

## Program Interruptions

The program interruption handler, DMSITP, receives control when a program interruption occurs. When DMSITP gets control, it stores the program old PSW and the contents of the registers 14, 15, 0, 1, and 2 into the program interruption element (PIE). (The routine that handles the SPIE macro instruction has already placed the address of the program interruption control area (PICA) into PIE.) DMSITP then determines whether or not the event that caused the interruption was one of those selected by a SPIE macro instruction. If it was not, DMSITP passes control to the DMSABN abend recovery routine.

If the cause of the interruption was one of those selected in a SPIE macro instruction, DMSITP picks up the exit routine address from the PICA and passes control to the exit routine. Upon return from the exit routine, DMSITP returns to the interrupted program by loading the original program check old PSW. The address field of the PSW was modified by a SPIE exit routine in the PIE.

## External Interruptions

An external interruption causes control to be passed to the external interrupt handler DMSITE.

If CMS IUCV support is active in the virtual machine and an IUCV external interrupt occurs, control is passed to the user exit specified on the HNDIUCV or CMSIUCV macro. If the user has issued the HNDEXT macro to trap external interrupts, DMSITE passes control to the user's exit routine. If the interrupt was caused by the timer, DMSITE resets the timer and types the BLIP character at the terminal. The standard BLIP timer setting is two seconds, and the standard BLIP character is uppercase, followed by the lowercase (it moves the typeball without printing). Otherwise, control is passed to the DEBUG routine.

## Machine Check Interruptions

Hard machine check interruptions on the real processor are not reflected to a CMS virtual user by CP. A message prints on the console indicating the failure. The user is then disabled and must IPL CMS again in order to continue.

# Functional Information

The most important thing to remember about CMS, from a debugging standpoint, is that it is a one-user system. The supervisor manages only one user and keeps track of only one user's file and storage chains. Thus, everything in a dump of a particular machine relates only to that virtual machine's activity.

You should be familiar with register usage, save area structuring, and control block relationships before attempting to debug or alter CMS.

## Register Usage

When a CMS routine is called, R1 must point to a valid parameter list (PLIST) for that program. On return, R0 may or may not contain meaningful information. For example, on return from a call to FILEDEF with no change, R0 contains a negative address if a new FCB (File Control Block) has been set up; otherwise, a positive address of the already existing FCB. R15 contains the return code, if any. The use of Registers 0 and 2 through 11 varies.

On entry to a command or routine called by SVC 202 the following are in effect:

| *Register* | *Contents* |
|---|---|
| 1 | The address of the PLIST supplied by the caller. |
| 12 | The address entry point of the called routine. |
| 13 | The address of a work area (12 doublewords) supplied by SVCINT. |
| 14 | The return address to the SVCINT routine. |
| 15 | The entry point (same as register 12). |

On return from a routine, Register 15 contains:

| *Return Code* | *Meaning* |
|---|---|
| 0 | No error occurred |
| <0 | Called routine not found |
| >0 | Error occurred |

If a CMS routine is called by an SVC 202, registers 0 through 14 are saved and restored by CMS.

Most CMS routines use register 12 as a base register.

## Structure of DMSNUC

DMSNUC is the portion of storage in a CMS virtual machine that contains system control blocks, flags, constants, and pointers.

The CSECTs in DMSNUC contain only symbolic references. This means that an update or modification to CMS, which changes a CSECT in DMSNUC, does not automatically force all CMS modules to be recompiled. Only those modules that refer to the area that was redefined must be recompiled.

### *USERSECT (User Area)*

The USERSECT CSECT defines space that is not used by CMS. A modification or update to CMS can use the 18 fullwords defined for USERSECT. There is a pointer (AUSER) in the NUCON area to the user space.

## *DEVTAB (Device Table)*

The DEVTAB CSECT is a table describing the devices available for the CMS system. The table contains the following entries:

- 1 console
- 26 disks
- 1 reader
- 1 punch
- 1 printer
- 4 tapes

You can change some existing entries in DEVTAB. Each device table entry contains the following information:

- Virtual device address
- Device flags
- Device types
- Symbol device name
- Address of the interrupt processing routine (for the console)

The virtual address of the console is defined at logon time. The symbolic names of the user disks can be altered dynamically with the ACCESS command. Figure 34 describes the devices supported by CMS.

| Virtual IBM Device Type | Virtual Address[1] | Symbolic Name (default) | Device Use |
|---|---|---|---|
| 3210, 3215, 1052, 3066, 3270 | cuu[2] | CON1 | System console |

**Figure 34 (Part 1 of 2). Devices Supported by a CMS Virtual Machine**

| Virtual IBM Device Type | Virtual Address[1] | Symbolic Name (default) | Device Use |
|---|---|---|---|
| 2314, 2319, 3310, 3330, 3340, 3350, 3370, 3375, 3380 | 190 | DSK0 | CMS System disk (read-only) |
| | 191[3] | DSK1 | Primary disk (user files) |
| | cuu | DSK2 | Minidisk (user files) |
| | cuu | DSK3 | Minidisk (user files) |
| | 192 | DSK4 | Minidisk (user files) |
| | cuu | DSK5 | Minidisk (user files) |
| | cuu | DSK6 | Minidisk (user files) |
| | cuu | DSK7 | Minidisk (user files) |
| | 19E | DSK8 | Minidisk (user files) |
| | cuu | DSK9 | Minidisk (user files) |
| | cuu | DSKH | Minidisk (user files) |
| | cuu | DSKI | Minidisk (user files) |
| | cuu | DSKJ | Minidisk (user files) |
| | cuu | DSKK | Minidisk (user files) |
| | cuu | DSKL | Minidisk (user files) |
| | cuu | DSKM | Minidisk (user files) |
| | cuu | DSKN | Minidisk (user files) |
| | cuu | DSKO | Minidisk (user files) |
| | cuu | DSKP | Minidisk (user files) |
| | cuu | DSKQ | Minidisk (user files) |
| | cuu | DSKR | Minidisk (user files) |
| | cuu | DSKT | Minidisk (user files) |
| | cuu | DSKU | Minidisk (user files) |
| | cuu | DSKV | Minidisk (user files) |
| | cuu | DSKW | Minidisk (user files) |
| | cuu | DSKX | Minidisk (user files) |
| 2540, 2501, 3505 | 00C | RDR1 | Virtual reader |
| 2540, 3525 | 00D | PCH1 | Virtual punch |
| 1403, 1443, 3203, 3211, 3262, 3800, 4245, 3289-4 | 00E | PRN1 | Line printer |
| 2401, 2402, 2403, 2415, 2420, 3410, 3411, 3420, 3430, 8809 | 181-4 | TAP1-TAP4 | Tape drives |

**Figure 34 (Part 2 of 2). Devices Supported by a CMS Virtual Machine**

[1]The device addresses shown are those that are preassembled into the CMS resident device table. These need only be modified and a new device table made resident to change the addresses.

[2]The virtual address of the system console may be any valid multiplexer address.

[3]191 is the default user-accessed A-disk unless it is dynamically changed by an ACCESS at CMS initial program load (IPL).

## Structure of CMS Storage

Figure 35 , Figure 36 , and Figure 37 describe how CMS uses its virtual storage. The pointers indicated (MAINSTRT, MAINHIGH, and FREELOWE) are all found in NUCON (the nucleus constant area).

The sections of CMS storage have the following uses:

- *DMSNUC (X'00000' to X'05000').* This area contains pointers, flags, and other data updated by the various system routines.

- *Low-Storage DMSFREE User Free Storage Area ( X'05000' to X'0E000').* This area is a free storage area, from which user requests to DMSFREE are allocated.

- *Transient Program Area (X'0E000' to X'10000').* Since it is not essential to keep all nucleus functions resident in storage all the time, some of them are made "transient". This means that when they are needed, they are loaded from the disk into the transient program area. Such programs may not be longer than two pages, because that is the size of the transient area. (A page is 4096 bytes of virtual storage.) All transient routines must be serially reusable since they are not read in each time they are needed.

- *Low-Storage DMSFREE Nucleus Free Storage Area (X'10000' to X'20000').* This area is a free storage area from which nucleus requests to DMSFREE are allocated. The top part of this area contains the dummy hyperblocks for the "S" and "Y" disk with each block 48 bytes long. This area may be followed by the file status tables for the "S2" filemode files of the system disk and/or the "Y2" filemode files of the system disk extension. Note that if the system disk is formatted as 512, 1K, 2K, or 4K blocks, each FST is 64 bytes (X'40') long, and holds approximately 318 FSTs. If the system disk is formatted for 800 byte blocks, each FST is 40 bytes (X'28') long and holds approximately 509 FSTs. If there is enough room, the FREETAB table also occupies this area, just below the file status tables, if they are there. Each entry in the FREETAB table is one byte long and each byte represents one page (4K or 4096 bytes) of defined storage.

- *User Program Area (X'20000' to Loader Tables or CMS Nucleus, whichever has the lower value).* User programs are loaded into this area by the LOAD command. Storage allocated by means of the GETMAIN macro instruction is taken from this area, starting from the high address of the user program. In addition, this storage area can be allocated from the top down by DMSFREE, if there is not enough storage available in the low DMSFREE storage area. Thus, the usable size of the user program area is reduced by the amount of free storage that has been allocated from it by DMSFREE.

- *Loader Tables (top pages of storage).* The top of storage is occupied by the loader tables, which are required by the CMS loader. These tables indicate which modules are currently loaded in the user program area (and the transient program area after a LOAD command). The size of the loader tables can be varied by the SET LDRTBLS command. However, to successfully change the size of the loader tables, the SET LDRTBLS command must be issued immediately after IPL.

- *CMS Nucleus (suggested location: X'700000' to 'X'MB).* Segments 29, 30 and 31 of storage contain the reentrant code for the CMS Nucleus routines, shared copies of the system S-STAT and Y-STAT, and the S-disk and Y-disk FST tables. If there is not sufficient room to contain these tables in this area, they are placed in low-storage DMSFREE Nucleus free storage area. In shared CMS systems, these are the "protected segments," which must consist only of reentrant code, and may not be modified under any circumstances.

If the size of the user's virtual machine is defined below the end of the CMS nucleus (refer to label NUCSIGMA in Figure 35 CMS Storage Map 1), it is not possible to IPL by device name. This is because the CMS nucleus is too large to be loaded into the user's virtual storage. Therefore, the user can only IPL by system name (e. g. IPL CMS). The loader table is placed immediately below the CMS nucleus.

On the other hand, if the size of the user's virtual machine is defined above the ending location of the CMS nucleus (refer to Figure 36 CMS Storage Map 2 and Figure 37 CMS Storage Map 3), the user may IPL by either device name or system name.

IPLing by device name:

The S and Y-STAT, and the loader table are placed above the CMS nucleus. If there isn't enough room to contain the S and Y-STAT, they are placed in low storage. Likewise, if there is insufficient room for the loader table above the CMS nucleus (NUCSIGMA), it is placed below the nucleus. Any leftover free space above the nucleus is placed on the high DMSFREE chain.

IPLing by system name:

The shared copy of the S and Y-STAT and nucleus is used. The loader table is placed above the S and Y-STAT (NUCOMEGA) if there is sufficient room. If there is insufficient room to place the loader table above the S and Y-STAT, it is placed below the nucleus. Any leftover free space above the S and Y-STAT (NUCOMEGA) is placed on the high DMSFREE chain.

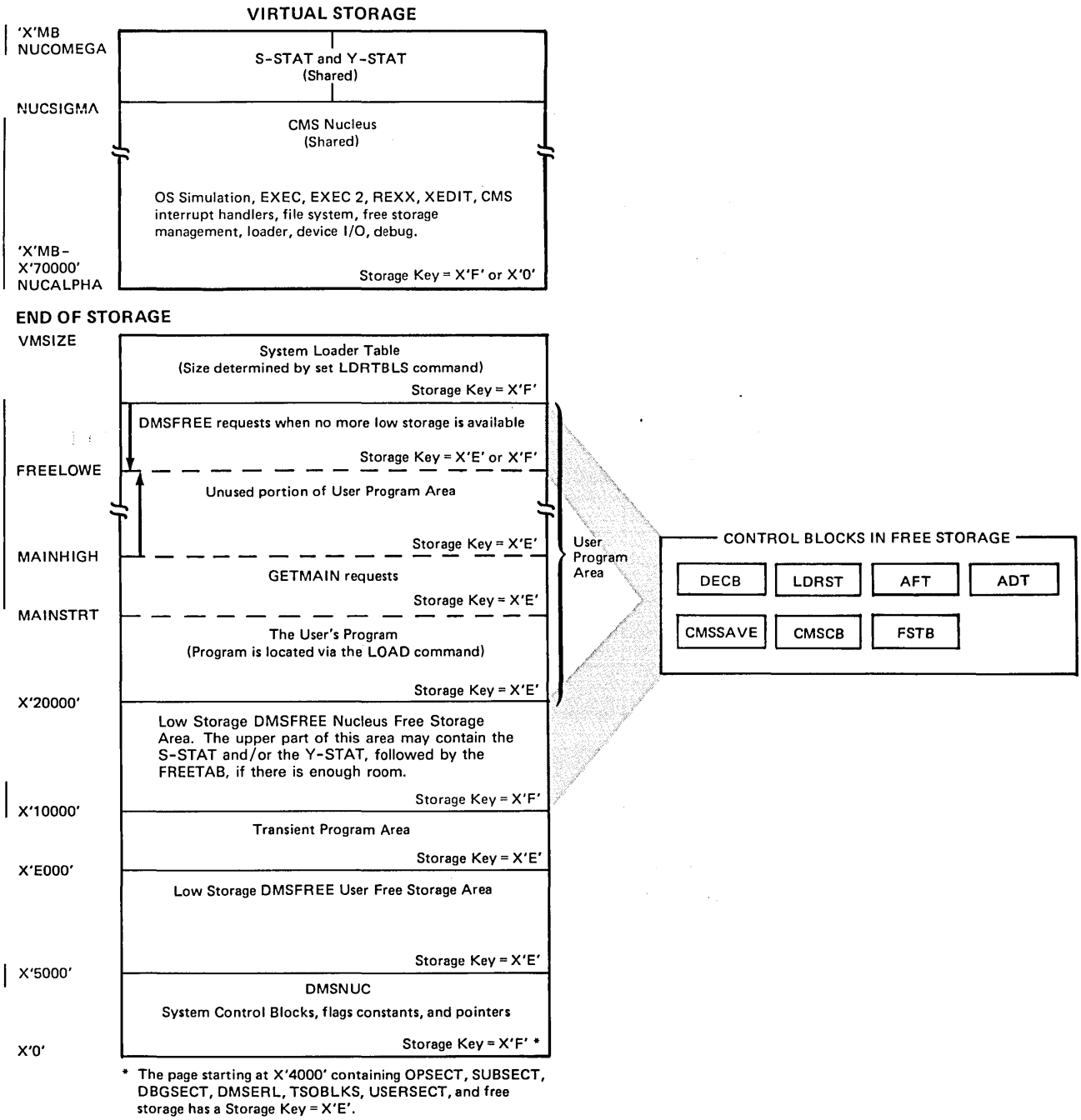



**Figure 35. CMS Storage Map 1.** CMS virtual storage usage when the CMS nucleus is larger than the user's virtual storage. In this case, you must IPL by system name (VMSIZE is less than NUCSIGMA).

**Note:** MAINHIGH is extended upward and FREELOWE is extended downward.

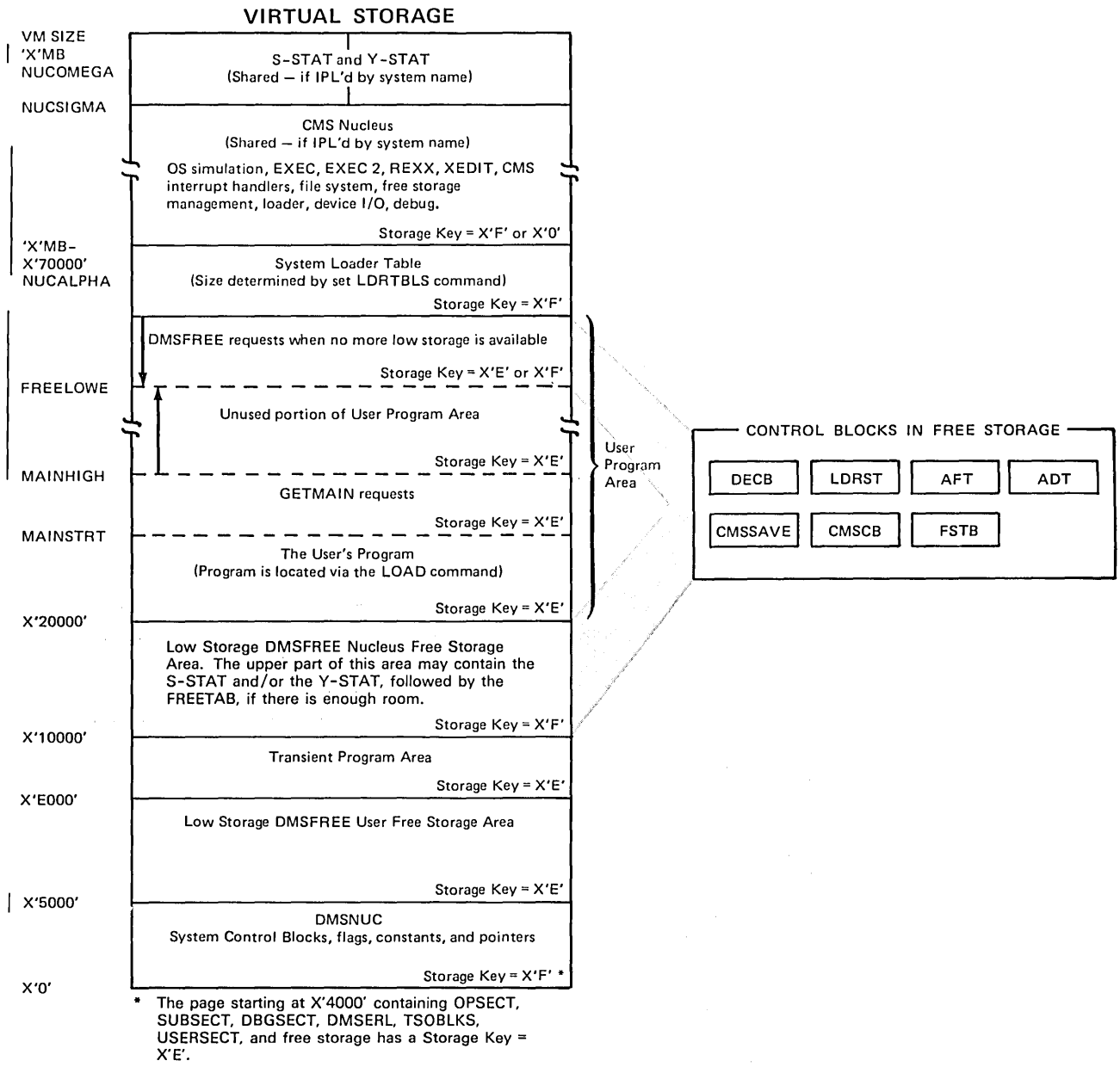
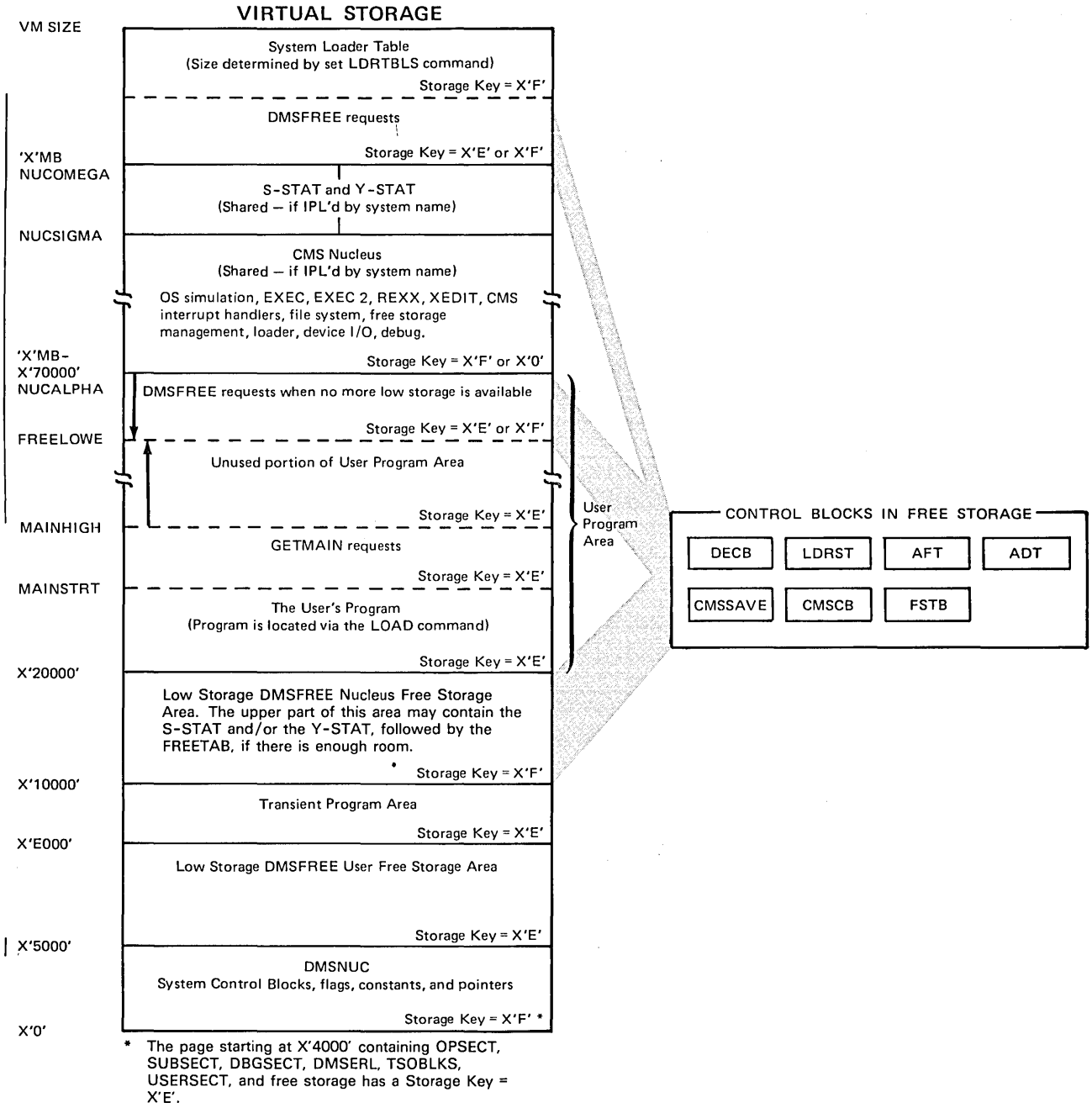



**Figure 36. CMS Storage Map 2.** Virtual storage usage when the user's virtual storage is larger than the CMS nucleus. The user may IPL by system name or device. In addition, this figure shows the case where there is insufficient room to place the loader table above S-STAT and Y-STAT.

**Note:** MAINHIGH is extended upward and FREELOWE is extended downward.




* The page starting at X'4000' containing OPSECT, SUBSECT, DBGSECT, DMSERL, TSOBLKS, USERSECT, and free storage has a Storage Key = X'E'.

**Figure 37. CMS Storage Map 3.** CMS Virtual storage usage when the user's virtual storage is larger than the CMS nucleus. The user may IPL by system name or device. In addition, this figure shows the case where there is sufficient room to place the loader table above S-STAT and Y-STAT.

**Note:** MAINHIGH is extended upward and FREELOWE is extended downward.

## Free Storage Management

Free storage can be allocated by issuing the GETMAIN or DMSFREE macros. Storage allocated by the GETMAIN macro is taken from the user program area, beginning after the high address of the user program.

Storage allocated by the DMSFREE macro can be taken from several areas.

If possible, DMSFREE requests are allocated from the low address free storage area. Otherwise, DMSFREE requests are satisfied from the storage above the user program area.

There are two types of DMSFREE requests for free storage: requests for USER storage and NUCLEUS storage. Because these two types of storage are kept in separate 4K pages, it is possible for storage of one type to be available in low storage, while no storage of the other type is available.

### *GETMAIN Free Storage Management*

All GETMAIN storage is allocated in the user program area, starting after the end of the user's actual program. Allocation begins at the location pointed to by the NUCON pointer MAINSTRT. The location MAINHIGH in NUCON is the "high extend" pointer for GETMAIN storage.

The STRINIT function initializes pointers used by CMS for simulation of OS GETMAIN/FREEMAIN storage management. In the usual CMS environment, that is, when execution is initiated by the LOAD and START commands, CMS calls the STRINIT macro as part of the LOAD preparation for execution. In an OS environment established by CMS, such as OSRUN, the STRINIT function has been performed by CMS and should not be done by the user program. In any case, the STRINIT macro should be issued only once in the OS environment, preceding the initial GETMAIN request. The format of the STRINIT macro is:

| | | |
|---|---|---|
| [label] | STRINIT | [TYPCALL=[SVC \| BALR]] |

*where:*

TYPCALL=[SVC | BALR]

indicates how control is passed to DMSSTG, the routine that processes the STRINIT macro. Since DMSSTG is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR) while routines that are not nucleus-resident must use linkage SVC (TYPCALL=SVC). If no operands are specified, the default is TYPECALL=SVC.

When the STRINIT macro is executed, both MAINSTRT and MAINHIGH are initialized to the end of the user's program, in the user program area. The end of the user's program is the upper boundary of the load module created by the CMS LOAD and INCLUDE commands. This upper boundary value is stored in the NUCON field LOCCNT. When execution of the user's program is started and the

and MAINHIGH. During execution of the user's program the LOCCNT field is used within CMS to pass starting and ending addresses of files loaded by OS simulation (see Notes below). As storage is allocated from the user program area to satisfy GETMAIN requests, the MAINHIGH pointer is adjusted upward. Such adjustments are always in multiples of doublewords, so that this pointer is always on a doubleheader boundary. As the allocated storage is returned, the MAINHIGH pointer is adjusted downward.

The pointer MAINHIGH can never be higher than FREELOWE, the "low extend" pointer for DMSFREE storage allocated in the user program area. If a GETMAIN request cannot be satisfied without extending MAINHIGH above FREELOWE, then GETMAIN takes an error exit, indicating that insufficient storage is available to satisfy the request.

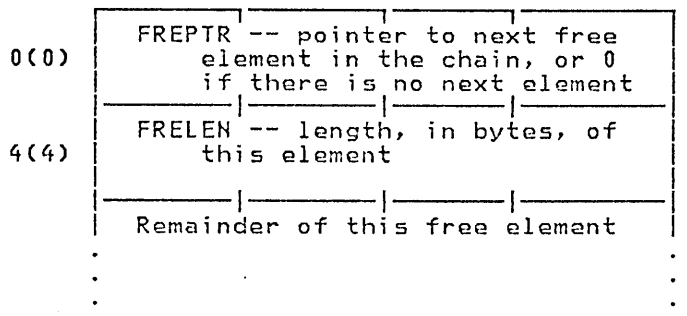
The area between MAINSTRT and MAINHIGH may contain blocks of storage that are not allocated and that are, therefore, available for allocation by a GETMAIN instruction. These blocks are chained together, with the first one pointed to by the NUCON location MAINLIST. Refer to Figure 35, Figure 36 , and Figure 37 for a description of CMS virtual storage usage.

**Notes:**

1. Reissuing the STRINIT macro during execution of an OS program, or issuing the STRINIT macro without having done a CMS LOAD is not advised because the value in LOCCNT will not have been appropriately set, possibly causing a subsequent storage management failure.

2. A high level language may issue a STRINIT. In this case, a user should not issue an additional STRINIT.

The format of an element on the GETMAIN free element chain is as follows:




When CMS issues the GETMAIN macro instruction for a variable amount of storage, the following formula determines the amount of storage obtained:

Amount of storage = 6 pages + 1 additional page for each 256K bytes obtained in excess of 512K bytes.

## *DMSFREE Free Storage Management*

The DMSFREE macro allocates CMS free storage. The format of the DMSFREE macro is:

| | | |
|---|---|---|
| [label] | DMSFREE | DWORDS= {n / (0)} [,MIN= {n / (1)}]<br>[,TYPE=[USER / NUCLEUS]] [,ERR= [laddr / *]]<br>[,AREA=[LOW / HIGH]] [,TYPCALL=[SVC / BALR]] |

*where:*

label
is any valid assembler language label.

DWORDS= {n / (0)}
is the number of doublewords of free storage requested. DWORDS=n specifies the number of doublewords directly and DWORDS=(0) indicates that register 0 contains the number of doublewords requested. Do not specify any register other than register 0.

CMS returns, in register 0, the number of doublewords allocated and, in register 1, the address of the first byte of allocated storage.

MIN= {n / (1)}
indicates a variable request for free storage. If the exact number of doublewords indicated by DWORDS operand is not available, then the largest block of storage that is greater than or equal to the minimum is requested. MIN=n specifies the minimum number of doublewords of free storage directly while MIN=(1) indicates that the minimum is in register 1. Do not specify any register other than register 1.

TYPE=[USER / NUCLEUS]
indicates the type of CMS storage with which this request for free storage is filled: USER or NUCLEUS.

ERR=[laddr / *]
is the return address if any error occurs. "laddr" is any address that can be referred to in an LA (load address) instruction. The error return is taken if there is a macro coding error or if there is no enough free storage available to fill the request. If the asterisk (*) is specified for the return address, the error return is the same as a normal return. There is no default for this operand. If it is omitted and an error occurs, the system abends.

AREA=[LOW / HIGH]
indicates the area of CMS free storage from which this request for free storage is filled. LOW indicates any free storage below the user areas,

depending on the storage requested. HIGH indicates DMSFREE storage above the user area. If AREA is not specified, storage is allocated wherever it is available.

TYPCALL=[SVC / BALR]
indicates how control is passed to DMSFREE. Because DMSFREE is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR) while routines that are not nucleus-resident must use linkage SVC (TYPCALL=SVC).

The FREELOWE pointer in NUCON indicates the amount of storage that DMSFREE has allocated from the high portion of the user program area. These pointers are initialized to the beginning of the loader tables.

The pointer FREELOWE is the "low extend" pointer of DMSFREE storage in the user program area. As storage is allocated from the user program area to satisfy DMSFREE requests, this pointer is adjusted downward. Such adjustments are always in multiples of 4K bytes, so that this pointer is always on a 4K boundary. As the allocated storage is returned, this pointer is adjusted upward and the freed pages are released by issuing a DIAGNOSE X'10' to CP.

The pointer FREELOWE can never be lower than MAINHIGH, the "high extend" pointer for GETMAIN storage. If a DMSFREE request cannot be satisfied without extending FREELOWE below MAINHIGH, then DMSFREE takes an error exit, indicating that storage is insufficient to satisfy the request. Figure 35 on page 320 , Figure 36 on page 321, and Figure 37 on page 322 show the relationship of these storage areas.

The FREETAB free storage table is usually kept in nucleus low FREE storage. However, the FREETAB may be located at the top of the user program area. This table contains a code indicating the use of that page of virtual storage. The codes in this table are as follows:

USERCODE (X'01') The page is assigned to user storage.

NUCCODE (X'02') The page is assigned to nucleus storage.

TRNCODE (X'03') The page is part of the transient program area.

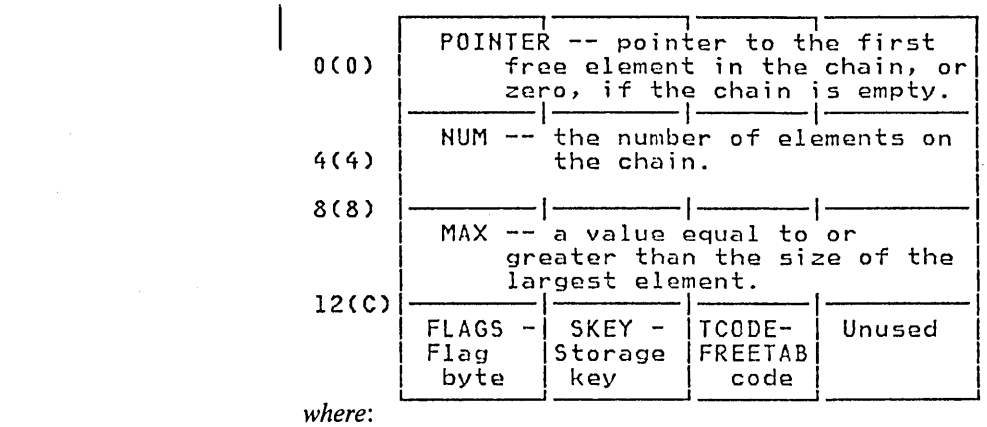
USARCODE (X'04') The page is an unassigned page in the user program area.

SYSCODE (X'05') The page is none of the above. The page is assigned to system storage, system code, or the loader tables.

Other DMSFREE storage pointers are maintained in the DMSFRT CSECT, in NUCON. The four chain header blocks are the most important fields in DMSFRT. The four chains of unallocated elements are:

- The low storage nucleus chain
- The low storage user chain
- The high storage nucleus chain
- The high storage user chain

For each of these chains of unallocated elements, there is a control block consisting of four words, with the following format:




*where*:

POINTER points to the first element on this chain of free elements. If there are no elements on this free chain, then the POINTER field contains all zeros.

NUM contains the number of elements on this chain of free elements. If there are no elements on this free chain, then this field contains all zeros.

MAX is used to avoid searches that will fail. It contains a number not exceeding the size, in bytes, of the largest element on the free chain. Thus, a search for an element of a given size is not made if that size exceeds the MAX field. However, this number may actually be larger than the size of the largest free element on the chain.

FLAGS The following flags are used:

FLCLN (X'80') -- Clean-up flag. This flag is set if the chain must be updated. This is necessary in the following circumstances:

- If one of he two high storage chains contains a 4K page to which FREELOWE points, then that page can be removed from the chain, and FREELOWE can be increased.
- All completely unallocated 4K pages are kept on the user chain, by convention. Thus, if one of the nucleus chains (low storage or high storage) contains a full page, then this page must be transferred to the corresponding user chain.

FLCLB (X'40') -- Destroyed flag. Set if the chain has been destroyed.

FLHC (X'20') -- High storage chain. Set for both the nucleus and user high storage chains.

FLUN (X'10') -- Nucleus chain. Set for both the low storage and high storage chains.

FLPA (X'08') -- Page available. This flag is set if there is a full 4K page available on the chain. This flag may be set even if there is no such page available.

SKEY contains the one-byte storage key assigned to storage on this chain.

TCODE contains the one-byte FREETAB table code for storage on this chain.

### Allocating User Free Storage

When DMSFREE with TYPE=USER (the default) is called, one or more of the following steps are taken in an attempt to satisfy the request. As soon as one of the following steps succeeds, then the user free storage allocation processing terminates.

1. Search the low storage chain for a block of the required size.
2. Search the high storage user chain for a block of the required size.
3. Extend high storage user storage downward into the user program area, modifying FREELOWE in the process.
4. For variable request, put all available storage in the user program area into the high storage user chain, and then allocate the largest block available on either the high storage user chain or the low storage user chain. The allocated block is not satisfactory unless it is larger than the minimum requested size.

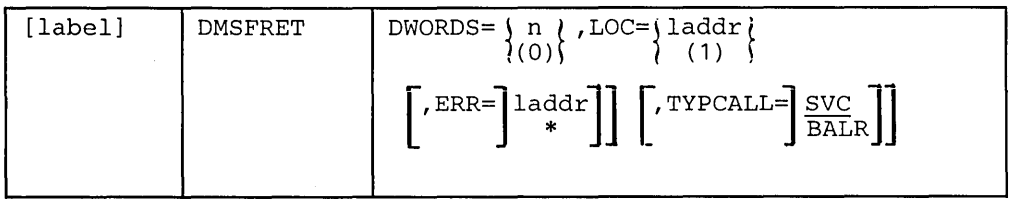
### Allocating Nucleus Free Storage

When DMSFREE with TYPE=NUCLEUS is called, the following steps are taken in an attempt to satisfy the request, until one succeeds:

1. Search the low storage nucleus chain for a block of the required size.
2. Search the high storage nucleus chain for a block of the required size.
3. Get free pages from the high storage user chain, if they are available, and put them on the high storage nucleus chain.
4. Extend high storage nucleus downward into the User Program Area, modifying FREELOWE in the process.
5. For variable requests, put all available pages from the user chain and the user program area onto the high storage nucleus chain, and allocate the largest block available on either the low storage nucleus chain, or the high storage nucleus chain.

### Releasing Storage

The DMSFRET macro releases free storage previously allocated with the DMSFREE macro. The format of the DMSFRET macro is:




*where*:

`label`
is any valid assembler language label.

DWORDS={n / (0)}

is the number of doublewords of storage to be released. DWORDS=n specifies the number of doublewords directly and DWORDS=(0) indicates that register 0 contains the number of doublewords being released. Do not specify any register other than register 0.

LOC={laddr / (1)}

is the address of the block of storage being released. "laddr" is any address that can be referred to in an LA (load address) instruction. LOC=laddr specifies the address directly while LOC=(1) indicates the address is in register 1. Do not specify any register other than register 1.

ERR=[laddr / *]

is the return address if any error occurs. "laddr" is any address that can be referred to in an LA (load address) instruction. The error return is taken if there is a macro coding error or if there is not enough free storage available to fill the request. If the asterisk (*) is specified for the return address, the error return is the same as a normal return. There is no default for this operand. If it is omitted and an error occurs, the system abends.

TYPCALL=[SVC / BALR]

indicates how control is passed to DMSFRET. Since DMSFRET is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR), while routines that are not nucleus-resident must use SVC linkage (TYPCALL=SVC).

When DMSFRET is called, the block being released is placed on the appropriate chain. At that point, the final update operation is performed, if necessary, to advance FREELOWE, or to move pages from the nucleus chain to the corresponding user chain.

Similar update operations are performed, when necessary, after calls to DMSFREE, as well.

## *Releasing Allocated Storage*

Storage allocated by the GETMAIN macro instruction may be released in either of the following ways:

1. A specific block of such storage may be released by means of the FREEMAIN macro instruction.

2. Whenever any user routine or CMS command abends (so that the routine DMSABN is entered), and the ABEND recovery facility of the system is invoked, all GETMAIN storage pointers are reset.

Storage allocated by the DMSFREE macro instruction may be released in either of the following ways:

1. A specific block of such storage may be released by means of the DMSFRET macro instruction.

2. Whenever any user routine or CMS command abnormally terminates (so that the routine DMSABN is entered), and the abend recovery facility of the system is invoked, all DMSFREE storage with TYPE=USER is released automatically.

Except in the case of abend recovery, storage allocated by the DMSFREE macro is never released automatically by the system. Thus, storage allocated by means of this macro instruction should always be released explicitly by means of the DMSFRET macro instruction.

## *DMSFRE Service Routines*

The DMSFRES macro instruction is used by the system to request certain free storage management services.

The format of the DMSFRES macro is:

| | | |
|---|---|---|
| [label] | DMSFRES | INIT1<br>INIT2 [,TYPCALL=[SVC / BALR]]<br>CHECK<br>CKON<br>CKOFF<br>UREC<br>CALOC |

*where*:

`label` is any valid Assembler language label.

`INIT1` invokes the first free storage initialization routines, so that free storage requests can be made to access the system disk. Before INIT1 is invoked, no free storage requests may be made. After INIT1 has been invoked, free storage requests may be made, but these are subject to the following restraints until the second free storage management initialization routine has been invoked:

- All requests for USER type storage are changed to requests for NUCLEUS type storage.
- Error checking is limited before initialization is complete. In particular, it is sometimes possible to release a block that was never allocated.
- All requests that are satisfied in high storage must be of a temporary nature, since all storage allocated in high storage is released when the second free storage initialization routine is invoked.

When CP's saved system facility is used, the CMS system is saved at the point just after the A-disk has been made accessible. It is necessary for DMSFRE to be used before the size of virtual storage is known, since the saved system can be used on any size virtual

machine. Thus, the first initialization routine initializes DMSFRE so that limited functions can be requested, while the second initialization routine performs the initialization necessary to allow the full functions of DMSFRE to be exercised.

INIT2 invokes the second initialization routine. This routine is invoked after the size of virtual storage is known, and it performs initialization necessary to allow all the functions of DMSFRE to be used. The second initialization routine performs the following steps:

- Releases all storage that has been allocated in the high storage area.
- Allocates the FREETAB free storage table. This table contains one byte for each 4K page of virtual storage, and so cannot be allocated until the size of virtual storage is known.
- The FREETAB table is initialized, and all storage protection keys are initialized.

CHECK invokes a routine that checks all free storage pointer chains for consistency and correctness. Thus, it checks to see whether or not any free storage pointers have been destroyed. The option can be used at any time for system debugging.

CKON turns on a flag that causes the CHECK routine to be invoked each time a call is made to DMSFREE or DMSFRET. This can be useful for debugging purposes (for example, when you wish to identify the routine that destroyed free storage management pointers). Care should be taken when using this option, since the CHECK routine is coded to be thorough rather than efficient. Thus, after the CKON option has been invoked, each call to DMSFREE or DMSFRET will take much longer to be completed than before.

CKOFF turns off the flag that was turned on by the CKON option.

UREC is used by DMSABN during the abend recovery process to release all user storage.

CALOC is used by DMSABN after the abend recovery process has been completed. It invokes a routine which returns, in register 0, the number of doublewords of free storage that have been allocated. This number is used by DMSABN to determine whether or not the abend recovery has been successful.

TYPCALL=[SVC / BALR]

indicates how control is passed to DMSFRES. Since DMSFRES is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR), while routines that are not nucleus-resident must use SVC linkage (TYPCALL=SVC).

### *Error Codes from DMSFREE, DMSFRES, and DMSFRET*

A nonzero return code upon return from DMSFREE, DMSFRES, or DMSFRET indicates that the request could not be satisfied. Register 15 contains this return code, indicating which error occurred. The following codes apply to the DMSFREE, DMSFRES, and DMSFRET macros.

| *Code* | *Error* |
|---|---|
| 1 | (DMSFREE) Insufficient storage space is available to satisfy the request for free storage. In the case of a variable request, even the minimum request could not be satisfied. |
| 2 | (DMSFREE or DMSFRET) User storage pointers destroyed. |
| 3 | (DMSFREE, DMSFRET, or DMSFRES) Nucleus storage pointers destroyed. |
| 4 | (DMSFREE) An invalid size was requested. This error is taken if the requested size is not greater than zero. In the case of variable requests, this error exit is taken if the minimum request is greater than the maximum request. (However, the latter error is not detected if DMSFREE is able to satisfy the maximum request.) |
| 5 | (DMSFRET) An invalid size was passed to the DMSFRET macro. This error exit is taken if the specified length is not positive. |
| 6 | (DMSFRET) The block of storage that is being released was never allocated by DMSFREE. Such an error is detected if one of the following errors is found:<br>• The block does not lie entirely inside either the low storage free storage area or the user program area between FREELOWE and FREEUPPR.<br>• The block crosses a page boundary that separates a page allocated for USER storage from a page allocated for NUCLEUS type storage.<br>• The block overlaps another block already on the free storage chain. |
| 7 | (DMSFRET) The address given for the block being released is not doubleword aligned. |
| 8 | (DMSFRES) An invalid request code was passed to the DMSFRES routine. Since all request codes are generated by the DMSFRES macro, this error code should never appear. |
| 9 | (DMSFREE, DMSFRET, or DMSFRES) Unexpected and unexplained error in the free storage management routine. |

## CMS Handling of PSW Keys

The CMS nucleus protection scheme protects the CMS nucleus from inadvertent destruction by a user program. This mechanism, however, does not prevent you from writing in system storage intentionally. Because you can execute privileged instructions, you can issue a LOAD PSW (LPSW) instruction and load any PSW key you wish. If this occurs, there is nothing to prevent your program from:

- Modifying nucleus code
- Modifying a table or constant area
- Losing files by modifying a CMS file directory

In general, user programs and disk-resident CMS commands are executed with a PSW key of X'E', while nucleus code is executed with a PSW key of X'0'.

There are, however, some exceptions to this rule. Certain disk-resident CMS commands run with a PSW key of X'0', since they have a constant need to modify nucleus pointers and storage. The nucleus routines called by the GET, PUT, READ, and WRITE macros run with a user PSW key of X'E', to increase efficiency.

Two macros, DMSKEY and DMSEXS, are available to any routine that wishes to change its PSW key for some special purpose.

## *The DMSKEY Macro*

The DMSKEY macro may be used to change the PSW key to the user value or the nucleus value. The format of the DMSKEY macro is:

| | | |
|---|---|---|
| [label] | DMSKEY | {NUCLEUS[,NOSTACK]\|<br>USER[,NOSTACK]\|<br>LASTUSER[,NOSTACK]\|<br>RESET} |

*where:*

NUCLEUS — causes the nucleus storage protection key to be placed in the PSW, and the old contents of the second byte of the PSW are saved in a stack. This option allows the program to store into system storage, which is ordinarily protected.

USER — causes the user storage protection key to be placed in the PSW, and the old contents of the second byte of the PSW are saved in a stack. This option prevents the program from inadvertently modifying nucleus storage, which is protected.

LASTUSER — The SVC handler traces back through its system save areas for the active user routine closest to the top of the stack. The storage key in effect for that routine is placed in the PSW. The old contents of the second byte of the PSW are saved in a stack. This option should be used only by system routines that should enter a user exit routine. (OS macro simulation routines use this option when they want to enter a user-supplied exit routine. The exit routine is entered with the PSW key of the last user routine on the SVC system save area stack.)

NOSTACK — This option may be used with any of the above options to prevent the system from saving the second byte of the current PSW in a stack. If this is done, then no DMSKEY RESET need be issued later.

RESET — The second byte of the PSW is changed to the value at the top of the DMSKEY stack, and removed from the stack. Thus, the effect of the last DMSKEY NUCLEUS, DMSKEY USER, or DMSKEY LASTUSER request is reversed. However, if the

NOSTACK option was specified on the DMSKEY macro, the RESET option should not be used. A DMSKEY RESET macro must be executed for each DMSKEY NUCLEUS, DMSKEY USER, or DMSKEY LASTUSER macro that was executed and that did not specify the NOSTACK option. Failure to observe this rule results in program abnormal termination. CMS requires that the DMSKEY stack be empty when a routine terminates.

**Note:** The DMSKEY key stack has a current maximum depth of seven for each routine. In this context, a "routine" is anything invoked by an SVC call.

### *The DMSEXS Macro*

The DMSEXS, "execute in system mode", macro allows a routine executed with a user PSW key, to execute a single instruction with a nucleus PSW key. The single instruction may be specified as the argument to the DMSEXS macro, and that instruction is executed with a nucleus PSW key. This macro can be used instead of two DMSKEY macros. The format of the DMSEXS macro is:

| [label] | DMSEXS | op-code,operands |
|---|---|---|

The op-code and the operands of the Basic Assembler Language instruction to be executed must be given as arguments to the DMSEXS macro.

For example, execution of the sequence,

```
USING  NUCON,0
DMSEXS OI,OSSFLAGS,COMPSWT
```

causes the OI instruction to be executed with a zero protect key in the PSW. This sequence turns on the COMPSWT flag in the nucleus. It is reset with

```
DMSEXS NI,OSSFLAGS,255-COMPSWT
```

The instruction to be executed may be an EX instruction.

**Note:** Programs that modify or manipulate bits in CMS control blocks, however, may hinder the operation of CMS, causing it to function ineffectively.

Register 1 cannot be used in any way in the instruction being executed.

Whenever possible, CMS commands are executed with a user protect key. This protects the CMS nucleus in cases where there is an error in the system command that would otherwise destroy the nucleus. If the command must execute a single instruction or small group of instructions that modify nucleus storage, then the DMSKEY or DMSEXS macros are used, so that the system PSW key is used for as short a period of time as is possible.

## CMS SVC Handling

DMSITS (INTSVC) is the CMS system SVC handling routine. The general operation of DMSITS is as follows:

1. The SVC new PSW (low storage location X'60') contains, in the address field, the address of DMSITS1. The DMSITS module is entered whenever a supervisor call is executed.

2. DMSITS allocates a system and user save area. The user save area is used as a register save area (or work area) by the called routine.

3. The called routine is called (via a LPSW or BALR).

4. Upon return from the called routine, the save areas are released.

5. Control is returned to the caller (the routine that originally made the SVC call).

## *SVC Types and Linkage Conventions*

SVC conventions are important to any discussion of CMS because the system is driven by SVCs (supervisor calls). SVCs 202 and 203 are the most common CMS SVCs.

### SVC 202

SVC 202 is used for calling nucleus-resident routines, and for calling routines written as commands (for example, disk resident modules). SVC 202 can also be used for calling nucleus extensions.

A typical coding sequence for an SVC 202 call is the following:

```
LA    R1,PLIST
SVC   202
DC    AL4(ERRADD)
```

The "DC AL4(address)" instruction following the SVC 202 is optional, and may be omitted if the programmer does not expect any errors to occur in the routine or command being called. If included, an error return is made to the address specified in the DC unless the address is equal to 1. If the address is 1, return is made to the next instruction after the "DC AL4(1)" instruction. DMSITS determines whether this DC was inserted by examining the next byte following the SVC call. A nonzero byte indicates an instruction, a zero value indicates that "DC AL4(address)" or "DC AL4(1)" follows.

If you want to ignore errors, you can use the following sequence:

```
LA    R1,PLIST
SVC   202
DC    AL4(1)
```

Whenever an SVC 202 is issued, the contents of general purpose register 0 and 1 (GPR0 and GPR1) are passed intact to the called routine. GPR1 must point to an eight-character string, which may be the start of a tokenized plist. This character string must contain the symbolic name of the routine or command being called. The SVC handler only examines the name and the high-order byte of GPR1. The called routine decides whether to use the extended PLIST or the tokenized PLIST by examining the high-order byte of GPR1.

**Note:** Although an extended PLIST is provided, the called routine might not be set up to use the extended PLIST.

The following values may be found in the high-order byte of register 1:

| Value | Meaning | CMS supplied extended PLIST pointer in register 0? |
|---|---|---|
| X'00' | The call did not originate from an EXEC file or a command typed at the terminal. | No |
| X'01' | The call is from an EXEC 2 exec or the System Product Interpreter when "ADDRESS COMMAND" is specified. | Yes |
| X'02' | See "Dynamic Linkage/SUBCOM" in this manual. | Yes |
| X'05' | Used by the System Product Interpreter for external function calls. | Yes |
| X'06' | The command was invoked as an immediate command. This setting should never occur with SVC 202. | Yes |
| X'0B' | The command was called as a result of its name being typed at the terminal, by the "CMDCALL" command to invoke the command from EXEC 2, or from a System Product Interpreter EXEC when "ADDRESS CMS" is specified. | Yes |
| X'0C' | The call is the result of a command invoked from an CMS EXEC file with "&CONTROL" set to something other than "NOMSG" or "MSG". | No |
| X'0D' | The call is the result of a command invoked from an CMS EXEC file with "&CONTROL MSG" in effect (indicates that messages are to be displayed at the terminal). | No |
| X'0E' | The call is the result of a command invoked from an CMS EXEC file with "&CONTROL NOMSG" in effect. | No |
| X'FE' | This is an end-of-command call from DMSINT (CMS console command handler). See the NUCEXT function in the *VM/SP CMS Command and Macro Reference* for further details. | No |
| X'FF' | This is a service call from DMSABN (ABEND) or from NUCXDROP. See the NUCEXT function in the *VM/SP CMS Command and Macro Reference* for details. | No |

*Tokenized PLIST*: For a tokenized parameter list, the symbolic name of the function being called (8 character string, padded with blank characters on the right if needed) is followed by extra arguments depending on the actual routine or command being called. These arguments must be "tokenized"; that is, every parenthesis is considered an individual argument, and each argument may have a maximum length of eight characters.

*Extended PLIST*: For an extended parameter plist, no restriction is put on the structure of the argument list passed to the called routine or command. Register 0 points to the following consecutive words:

```
(a)  DC A(CMDBEG)
(b)  DC A(ARGBEG)
(c)  DC A(ARGEND)
(d)  DC A(0)
```

where the first three addresses are defined as in the following example:

```
CMDBEG EQU *
       DC C'testprog'
ARGBEG EQU *
       DC C'(file 2)'
ARGEND EQU *
```

```
CMDBEG EQU *  -indicates the beginning of the command name.
ARGBEG EQU *  -indicates the beginning of the argument list.
ARGEND EQU *  -indicates the end of the argument list.
```

a. The first word is the beginning address of the command.

b. The second gives the beginning address of the argument list.

c. The third gives the address of the byte immediately following the end of the argument list.

d. The fourth word may be used to pass any additional information required by individual called programs. If not used to pass additional information, this word should be zero so that programs which can receive optional information via this word may detect that none is provided in this call.

**Notes:**

1. It is specifically allowed that these four words be moved to some location convenient for the command resolution routines, or convenient for some other program executed between the caller's SVC 202 and entry to the program for which the parameter list is intended. For this reason, the called program may not assume additional words follow word 4, or that the storage address of these 4 words bears any relationship to other data addresses.

2. For function calls in the System Product Interpreter, two additional words are available. See the *VM/SP System Product Interpreter Reference*, SC24-5239, for more information on function calls and the two additional words.

*SVC 203*

SVC 203 is called by CMS macros to perform various internal system functions. It is used to define SVC calls for which no parameter list is provided. For example, DMSFREE parameters are passed in registers 0 and 1.

A typical calling sequence for an SVC 203 call is as follows:

```
SVC   203
DC    H'code'
```

The halfword decimal code following the SVC 203 indicates the specific routine being called. DMSITS examines this halfword code, taking the absolute value of

the code by an LPR instruction. The first byte of the result is ignored, and the second byte of the resulting halfword is used as an index to a branch table. The address of the correct routine is loaded, and control is transferred to it.

It is possible for the address in the SVC 203 index table to be zero. In this case, the index entry contains an 8-byte routine or command name, which is handled in the same way as the 8-byte name passed in the parameter list to an SVC 202.

The programmer indicates an error return by the sign of the halfword code. If an error return is desired, then the code is negative. If the code is positive, then no error return is made. The sign of the halfword code has no effect on determining the routine that is to be called, since DMSITS takes the absolute value of the code to determine the routine called.

Since only the second byte of the absolute value of the code is examined by DMSITS, seven bits (bits 1-7) are available as flags or for other uses. Thus, for example, DMSFREE uses these seven bits to indicate such things as conditional requests and variable requests.

When an SVC 203 is invoked, DMSITS stores the halfword code into the NUCON location CODE203, so that the called routine can examine the seven bits made available to it.

All calls made by means of SVC 203 should be made by macros, with the macro expansion computing and specifying the correct halfword code.

### User-Handled SVCs

The programmer may use the HNDSVC macro to specify the address of a routine that will handle any SVC call other than for SVC 202 and SVC 203.

In this case, the linkage conventions are as required by the user-specified SVC-handling routine.

### OS and VSE Macro Simulation SVC Calls

CMS supports selected SVC calls generated by OS and VSE macros, by simulating the effect of these macro calls. DMSITS is the initial SVC interrupt handler. If the SET DOS command has been issued, a flag in NUCON indicates that VSE macro simulation is to be used. Control is then passed to DMSDOS. Otherwise, OS macro simulation is assumed and DMSITS passes control to the appropriate OS simulation routine.

*Invalid SVC Calls*

There are several types of invalid SVC calls recognized by DMSITS.

1. Invalid SVC number. If the SVC number does not fit into any of the four classes described above, then it is not handled by DMSITS. An appropriate error message is displayed at the terminal, and control is returned directly to the caller.

2. Invalid routine name in SVC 202 parameter list. If the routine named in the SVC 202 parameter list is invalid or cannot be found, DMSITS handles the situation in the same way as it handles an error return from a legitimate SVC routine. The error code is -3.

3. Invalid SVC 203 code. If an invalid code follows SVC 203 inline, then an error message is displayed, and the abend routine is called to terminate execution.

## *Search Hierarchy for SVC 202*

When a program issues SVC 202, passing a routine or command name in the parameter list, then DMSITS searches for the specified routine or command. (In the case of SVC 203 with a zero in the table entry for the specified index, the same logic must be applied.)

Figure 38 , Part 2, shows the search logic following and SVC 202 call.

The search algorithm is as follows:

1. A check is made to determine if the specified name is known dynamically to CMS through the SUBCOM function.

2. A check is made to see if the specified name is a nucleus extension routine. If this is the case, the control goes to the specified nucleus extension routine.

   **Note:** This step is skipped if the high-order byte of register 1 contains X'03' or X'04'. X'03' indicates that an extended plist is provided. X'04' indicates that a tokenized plist is provided. For both X'03' and X'04', values are translated to X'01' and X'00', respectively, by the SVC interrupt handler before the called program is entered.

3. A check is made to see if there is a routine with the specified name currently occupying the system transient area. If this is the case, then control is transferred there.

4. The system function name table is searched, to see if a command by this name is a nucleus-resident command. If the search is successful, control goes to the specified nucleus routine.

5. A search is then made for a disk file with the specified name as the filename, and MODULE as the filetype. The search is made in the standard disk search order. If this search is successful, then the specified module is loaded (via the LOADMOD command), and control passes to the storage location now occupied by the command.

6. If all searches so far have failed, then DMSINA (ABBREV) is called, to see if the specified routine name is a valid system abbreviation for a system command or function. User-defined abbreviations and synonyms are also checked. If this search is successful, then steps 2 through 5 are repeated with the full function name.

7. If all searches fail, then an error code of -3 is issued.

### Commands Entered from the Terminal

When a command is entered from the terminal, DMSINT processes the command line, and calls the scan routine to convert it into a parameter list consisting of eight-byte entries.

See Figure 38 for a description of this search for a command name. The following search is performed:

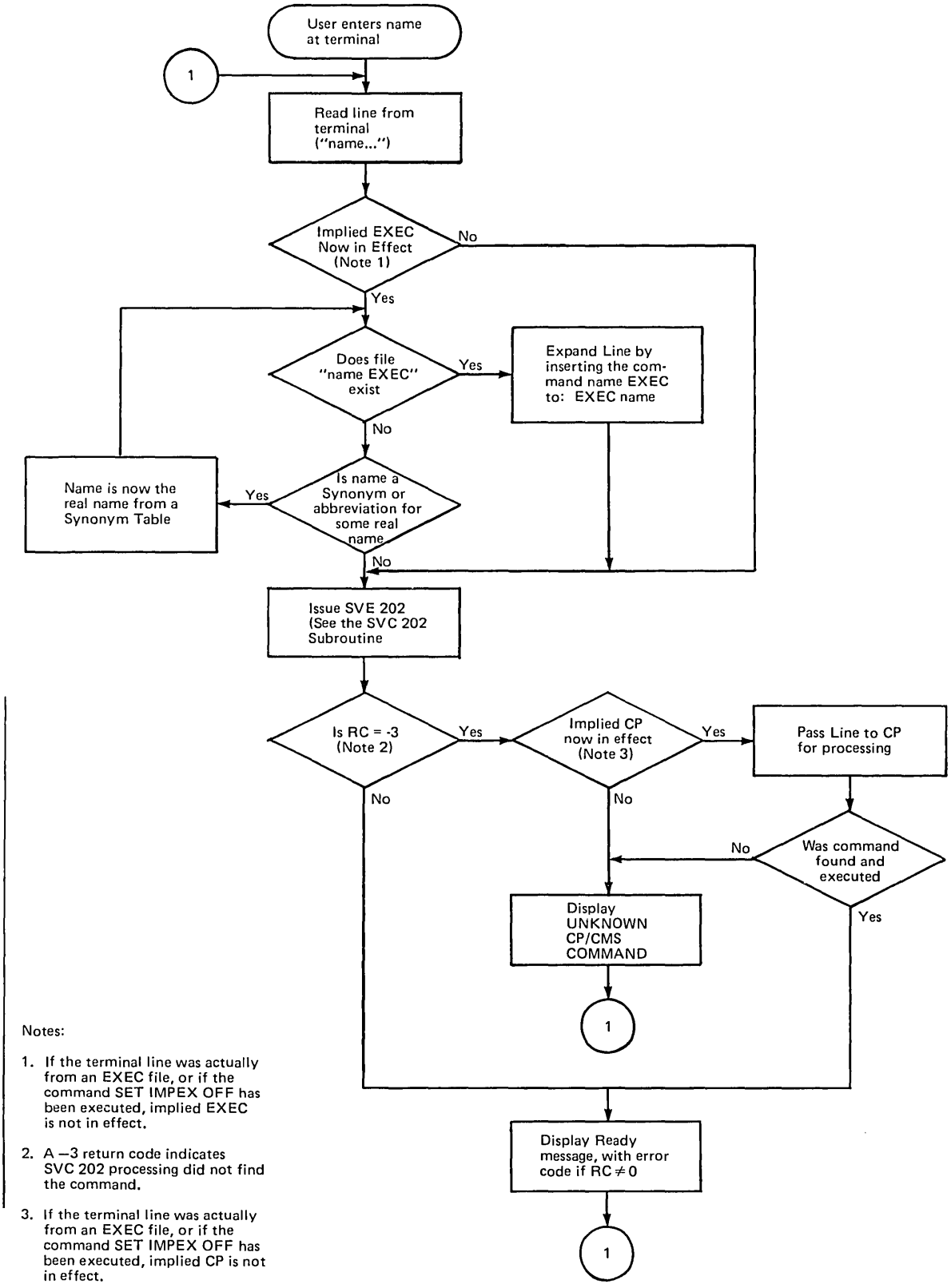
1. DMSINT searches for a disk file whose filename is the command name, and whose filetype is EXEC. If this search is successful, EXEC is invoked to process the EXEC file.

   If not found, the command name is considered to be an abbreviation and the appropriate tables are examined. If found, the abbreviation is replaced by its full equivalent and the search for an EXEC file is repeated.

2. If there is no EXEC file, DMSINT executes SVC 202, passing the scanned parameter list, with the command name in the first eight bytes. DMSITS performs the search described for SVC 202 in an effort to execute the command.

3. If DMSITS returns to DMSINT with a return code of -3, indicating that the search was unsuccessful, then DMSINT uses the CP DIAGNOSE facility to attempt to execute the command as a CP command.

4. If all of these searches fail, then DMSINT displays the error message UNKNOWN CP/CMS COMMAND.




Notes:

1. If the terminal line was actually from an EXEC file, or if the command SET IMPEX OFF has been executed, implied EXEC is not in effect.
2. A −3 return code indicates SVC 202 processing did not find the command.
3. If the terminal line was actually from an EXEC file, or if the command SET IMPEX OFF has been executed, implied CP is not in effect.

**Figure 38 (Part 1 of 2). CMS Command (and Request) Processing**

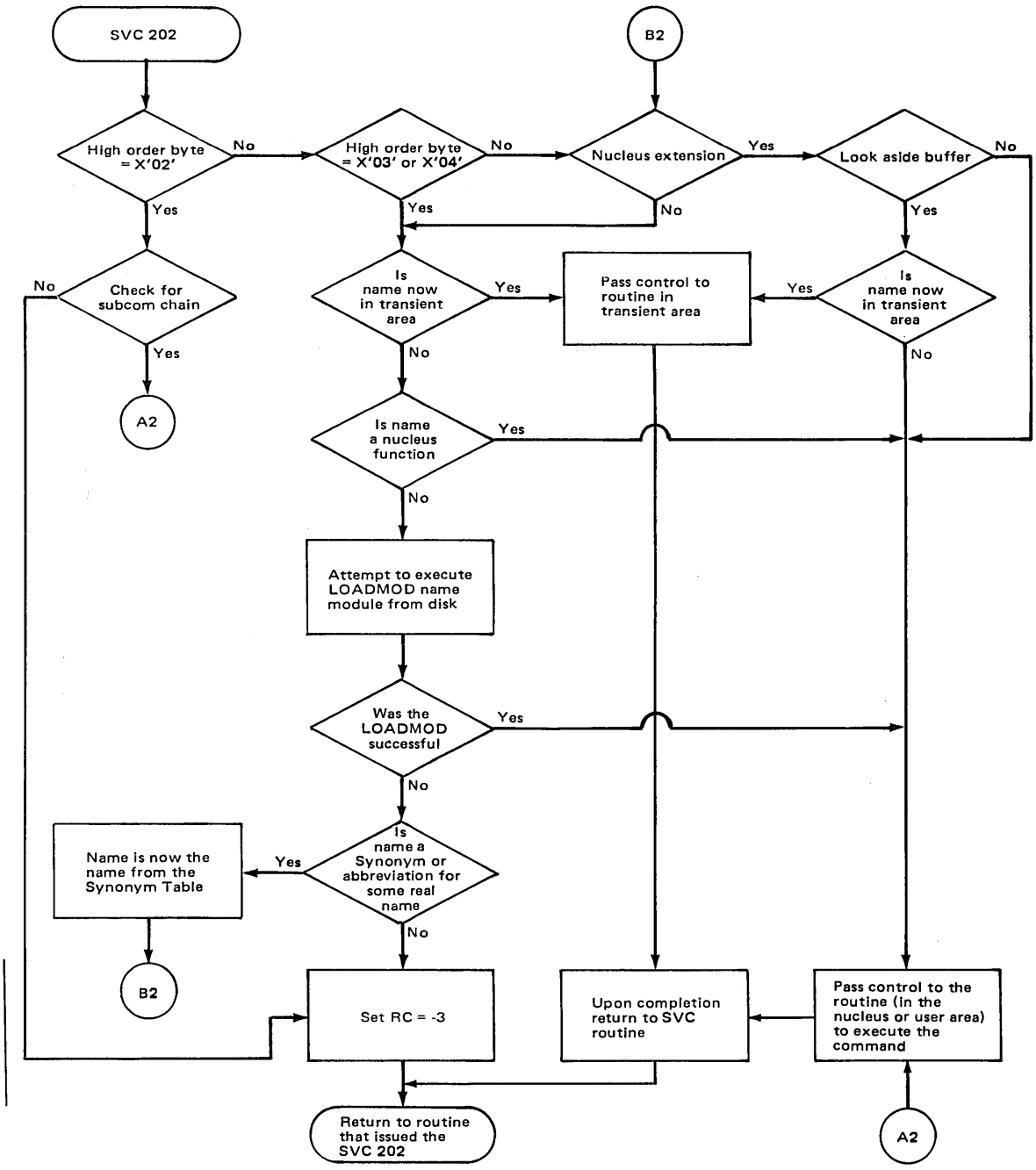



**Figure 38 (Part 2 of 2). CMS Command (and Request) Processing**

### *User and Transient Program Areas*

Two areas hold programs that are loaded from disk. These areas are called the user program area and the transient program area. (See Figure 35 , Figure 36 , and Figure 37 on page 322 for a description of CMS storage usage.)

The user program area starts at location X'20000' and extends upward to the loader tables. Generally, all user programs and certain system commands (such as EDIT, and COPYFILE) are executed in the user program area. Since only one program can be executing in the user program area at any one time, it is impossible (without unpredictable results) for one program being executed in the user program area to invoke, by means of SVC 202, a module that is also intended to be executed in the user program area.

The transient program area is two pages long, extending from location X'E000' to location X'FFFF'. It provides an area for system commands that may also be invoked from the user program area by means of an SVC 202 call. When a transient module is called by an SVC, it is normally executed with the PSW system mask disabled for I/O and external interrupts.

A program being executed in the transient program area may not invoke another program intended for execution in the transient program area.

DMSITS starts the programs to be executed in the user program area enabled for all interrupts but starts the programs to be executed in the transient program area disabled for all interrupts. The individual program may have to use the SSM (Set System Mask) instruction to change the current status of its system mask.

### *Called Routine Start-Up Table*

Figure 39 and Figure 40 on page 344 show how the PSW and registers are set up when the called routine is entered.

| "Called" Type | System Mask | Storage Key | Problem Bit |
|---|---|---|---|
| SVC 202 or 203 - Nucleus resident | Disabled | System | Off |
| SVC 202 - Nucleus Extension Module | User Defined | User Defined | Off |
| SVC 202 or 203 - Transient area MODULE | Disabled | User | Off |
| SVC 202 or 203 - User area | Enabled | User | Off |
| User-handled | Enabled | User | Off |
| OS - VSE Nucleus resident | Disabled | System | Off |

**Figure 39 (Part 1 of 2). PSW Fields When Called Routine Starts**

| "Called" Type | System Mask | Storage Key | Problem Bit |
|---|---|---|---|
| OS - VSE Transient area module | Disabled | System | Off |

**Figure 39 (Part 2 of 2). PSW Fields When Called Routine Starts**

| Type | Registers 0 - 1 | Registers 2 - 11 | Register 12 | Register 13 | Register 14 | Register 15 |
|---|---|---|---|---|---|---|
| SVC 202 or 203 | Same as caller | Unpredictable | Address of called routine | User save area | Return Address to DMSITS | Address of called routine |
| Other | Same as caller | Same as caller | Address of caller | User save area | Return address to DMSITS | Same as caller |

**Figure 40. Register Contents When Called Routine Starts**

## *Returning to the Calling Routine*

When the called routine finishes processing, control is returned to DMSITS, which in turn returns control to the calling routine.

### Return Location

The return is accomplished by loading the original SVC old PSW (which was saved at the time DMSITS was first entered), after possibly modifying the address field. The address field modification depends upon the type of SVC call, and upon whether or not the called routine indicated an error return.

For SVC 202 and 203, the called routine indicates a normal return by placing a zero in register 15 and an error return by placing a nonzero code in register 15. If the called routine indicates a normal return, then DMSITS makes a normal return to the calling routine. If the called routine indicates an error return, DMSITS passes the error return to the calling routine, if one was specified, and abnormally terminates if none was specified.

For an SVC 202 not followed by "DC AL4(address)" or "DC AL4(1)," a normal return is made to the instruction following the SVC instruction, and an error return causes an abend. For an SVC 202 followed by "DC AL4(address)", a normal return is made to the instruction following the DC, and an error return is made to the address specified in the DC, unless the address is equal to 1. If the address is 1, both normal and error, return is made to the next instruction after the "DC AL4(1)" instruction. In either case, register 15 contains the return code passed back by the called routine.

For an SVC 203 with a positive halfword code, a normal return is made to the instruction following the halfword code, and an error return causes an abend. For an SVC 203 with a negative halfword code, both normal and error returns are made to the instruction following the halfword code. In any case, register 15 contains the return code passed back by the called routine.

For macro simulation SVC calls, and for user-handled SVC calls, no error return is recognized by DMSITS. As a result, DMSITS always returns to the calling routine by loading the SVC old PSW, which was saved when DMSITS was first entered.

### Register Restoration

Upon entry to DMSITS, all registers are saved as they were when the SVC instruction was first executed. Upon exiting from DMSITS, all registers are restored from the area in which they were saved at entry.

The exception to this is register 15 in the case of SVC 202 and 203. Upon return to the calling routine, register 15 always contains the value that was in register 15 when the called routine returned to DMSITS after it had completed processing.

### Called Routine Modifications to System Area

If the called routine has system status, so that it runs with a PSW storage protect key of 0, then it may store new values into the System Save Area.

If the called routine wishes to modify the location to which control is to be returned, it must modify the following fields:

- For SVC 202 and 203, it must modify the NUMRET and ERRET (normal and error return address) fields.
- For other SVCs, it must modify the address field of OLDPSW.

To modify the registers that are to be returned to the calling routine, the fields EGPR1, EGPR2, through EGPR15 must be modified.

If this action is taken by the called routine, then the SVCTRACE facility may print misleading information, since SVCTRACE assumes that these fields are exactly as they were when DMSITS was first entered. Whenever an SVC call is made, DMSITS allocates two save areas for that particular SVC call. Save areas are allocated as needed. For each SVC call, a system and user save area are needed.

When the SVC-called routine returns, the save areas are not released, but are kept for the next SVC. At the completion of each command, all SVC save areas allocated by that command are released.

The System Save Area is used by DMSITS to save the value of the SVC old PSW at the time of the SVC call, the calling routine's registers at the time of the call, and any other necessary control information. Since SVC calls can be nested, there can be several of these save areas at one time. The system save area is allocated in protected free storage.

The user save area contains 12 doublewords (24 words), allocated in unprotected free storage. DMSITS does not use this area at all, but simply passes a pointer to this area (via register 13.) The called routine can use this area as a temporary work area, or as a register save area. There is one user save area for each system save area. The USAVEPTR field in the system save area points to the user save area.

The exact format of the system save area can be found in the *VM/SP Data Areas and Control Block Logic, Volume 2*. The most important fields, and their uses, are as follows:

| *Field* | *Usage* |
|---|---|
| CALLER | (Fullword) The address of the SVC instruction that resulted in this call. |
| CALLEE | (Doubleword) Eight-byte symbolic name of the called routine. For OS and user-handled SVC calls, this field contains a character string of the form SVC nnn, where nnn is the SVC number in decimal. |
| CODE | (Halfword) For SVC 203, this field contains the halfword code following the SVC instruction line. |
| OLDPSW | (Doubleword) The SVC old PSW at the time that DMSITS was entered. |
| NRMRET | (Fullword) The address of the calling routine to which control is to be passed in the case of a normal return from the called routine. |
| ERRET | (Fullword) The address of the calling routine to which control is to be passed in the case of an error return from the called routine. |
| EGPRS | (16 Fullwords, separately labeled EGPR0, EGPR1, EGPR2, EGPR3, ..., EGPR15) The entry registers. The contents of the general registers at entry to DMSITS are stored in these fields. |
| EFPRS | (4 Doublewords, separately labeled EFPR0, EFPR2, EFPR4, EFPR6) The entry floating-point registers. The contents of the floating-point registers at entry to DMSITS are stored in these fields. |
| SSAVENXT | (Fullword) The address of the next system save area in the chain. This points to the system save area that is being used, or will be used, for any SVC call nested in relation to the current one. |
| SSAVEPRV | (Fullword) The address of the previous system save area in the chain. This points to the system save area for the SVC call in relation to which the current call is nested. |
| USAVEPTR | (Fullword) Pointer to the user save area for this SVC call. |

## Dynamic Linkage--Subcom

It is possible for a program that has already been loaded from disk to become known by name to CMS for the duration of the current command; such a program thus can be called via SVC 202. In addition, a program that has become known dynamically can make other programs known dynamically (if the first program can supply the entry points of the other programs).

To become known dynamically to CMS, a program or routine invokes the create function of SUBCOM. To invoke SUBCOM, issue the following calling sequence from an assembler language program:

```
         LA   R1,PLIST
         SVC  202
         DC   AL4(ERROR)
         .
         .
         .
PLIST    DS   0F
         DC   CL8'SUBCOM'
SUBCNAME DC   CL8'name'      COMMAND NAME
SUBCPSW  DC   XL2'0000'      SYSTEM MASK, STORAGE KEY, ETC.
         DC   AL2(0)         RESERVED
SUBCADDR DC   A(*-*)         ENTRY ADDRESS, -1 FOR QUERY
         DC   A(0)           USER WORD
```

SUBCOM creates an SCBLOCK control block containing the information specified in the SUBCOM parameter list. SVC 202 uses this control block to locate the specified routine. The SUBCOM chain of SCBLOCKs is released at the completion of a command (that is, when CMS displays the Ready message). See *VM/SP Data Areas and Control Block Logic, Volume 2* for a description of the SCBLOCK control block.

When a program issues an SVC 202 call to a program that has become known to CMS via SUBCOM, it places X'02' in the high-order byte of register one. Control passes to the called program at the address specified by the called program when it invoked SUBCOM.

The PSW specifies the system mask, the PSW key to be used, the program mask (and initial condition code), and the starting address for execution. The problem-state bit and machine-check bit may be set. The machine-check bit has no effect in CMS under CP. The EC-mode bit and wait-state bit cannot be set; they are always forced to zero. Also, one 4-byte user-defined word can be associated with the SUBCOM entry point, and referred to when the entry point is subsequently called.

**Note:** When control passes to the specified entry point, the register contents are:

R2 Address of SCBLOCK for this entry point.

R12 Entry point address.

R13 24-word save area address.

R14 Return address (CMSRET).

R15 Entry point address.

You can also use SUBCOM to delete the potential linkage to a program or routine's SCBLOCK, or to determine if an SCBLOCK exists for a program or routine. To delete a program or routine's SCBLOCK, issue:

```
DC   CL8'SUBCOM'
DC   CL8'program or routine name'
DC   8X'00'
```

To determine if an SCBLOCK exists for a program or routine, issue:

```
DC   CL8'SUBCOM'
DC   CL8'program or routine name'
DC   A(0)    SCBLOCK addressed as a returned value
DC   4X'FF'
```

Note that if 'SUBCOM name' is called from an EXEC file, the Query PLIST is the form of PLIST which will be issued.

To query the chain anchor issue:

```
DC   CL8'SUBCOM'
DS   CL8                (contents not relevant)
DS   AL4                Will receive chain anchor
                        contents from NUCSCBLK.
DC   AL4(1)             Indicates request for anchor.
```

Note that the anchor will be equal to F'0' if there are no SCBLOCKs on the chain.

Return codes from SUBCOM are:

0 - Successful completion. A new SCBLOCK was created, the specified SCBLOCK was deleted, or the specified program or routine has an SCBLOCK.

1 - No SCBLOCK exists for the specified program or routine. This is the return code for a delete or a query.

25 - No more free storage available. SCBLOCK cannot be created for the specified program or routine.

**Note:** If you create SCBLOCKS for several programs or routines with the same name, they will all be remembered, but only the last one to be created will be used. A SUBCOM delete request for that name will eliminate only the most recently created SCBLOCK, making active the next most recently created SCBLOCK with the same name.

When control returns to CMS after a console input command has terminated, the entire SUBCOM chain SCBLOCKs is released; none of the subcommands established during that command are carried forward to be available during execution of the next console command.

## System Product Editor Interface to Access Files in Storage

CMS uses the SUBCOM facility to allow a number of CMS commands to use an XEDIT interface to access files in storage. Applications can read or write specific records without having to go to disk or use the program stack to transfer the data to or from XEDIT. This improves performance.

This interface is used internally by CMS for processing the FILELIST, HELP, PEEK, and SENDFILE commands. The interface is invoked by specifying the XEDIT option on the LISTFILE or NAMEFIND commands. This option may only be specified from the XEDIT environment.

When using this interface from an application program, only the extended Parameter List can be used, and the high-order byte of of Register 1 must contain X'02' to indicate SUBCOM is being used.

The application can invoke this interface via SVC 202 or via a BALR instruction. Because XEDIT is a nucleus-resident routine, other nucleus-resident routines can branch directly to it while routines that do not resides in the nucleus use SVC linkage. When using an SVC 202, register 1 must point to the FSCB where the name of the routine being invoked is the first token. The high-order byte of register 1

must also be X'02'. When using BALR, the calling program can determine the entry point it wants by using SUBCOM. In this case, register 1 points to the FSCB and register 2 points to the SCBLOCK. The address of the the SCBLOCK has been returned from SUBCOM.

The routines available, their entry point names, and error return codes are:

- DMSXFLST - This routine returns the characteristics of a file (RECFM, LRECL, etc). It also ensures that the file is in the XEDIT ring. The return codes are:

  0 File is in the XEDIT ring
  24 Incomplete fileid specified
  28 File is not in the XEDIT ring

  **Note:** Return codes are similar to those for ESTATE.

- DMSXFLRD - This routine transfers one record from XEDIT storage to the calling program. If RECFM=F, it may transfer more than one record. The return codes are:

  0 READ performed
  1 File is not in the XEDIT ring
  2 Invalid buffer address
  5 Number of items equals zero
  7 RECFM is not 'F' or 'V'
  8 Buffer is too small (Records truncated)
  11 Number of items is not equal to one for V-file
  12 End of file

  **Note:** Return codes are similar to those for FSREAD.

- DMSXFLWR - This routine transfers one record from the calling program to XEDIT storage. If RECFM=F, it may transfer more than one record. The return codes are:

  0 WRITE performed
  2 User buffer address equals zero
  7 Skip over unwritten records
  8 Number of bytes is not specified
  11 RECFM is not 'F' or 'V'
  13 No more space is available
  14 Number of bytes is not integrally divisible by the number of item
  15 Item length is not the same as previous
  16 RECFM of 'F' or 'V' is not the same as previous
  18 Number of items is not equal to one for V-file
  28 File is not in the XEDIT ring

  **Note:** Return codes are similar to those for FSWRITE.

- DMSXFLPT - This routine moves the current line pointer to a record specified by the calling program. The return codes are:

  0 POINT performed
  1 File not found
  2 Invalid FSCB

**Note:** Return codes are similar to those for FSPOINT.

When the interface is used, XEDIT determines if a file is in the XEDIT ring (active in storage) and does the processing required. The files in the XEDIT ring are always open. New files may be added to the ring with the XEDIT subcommand. Files in the ring may be closed with the FILE or QUIT subcommands.

The current line pointer serves the function of both the read and write pointers of the CMS file system. If RECNO=0 is specified in a call to DMSXFLRD, then the data will be transferred to the calling program starting at the current line pointer. Transfer is stopped when the specified number of lines has been transferred or when end-of-file is reached. The current line pointer is advanced by one for each record transferred to the calling program. If the current line pointer was at the end-of-file when DMSXFLRD was called, no data is transferred and an end-of-file condition is returned.

If RECNO=0 is specified in a call to DMSXFLWR, the new records are written starting at the line pointed to by the current line pointer, replacing any existing records, or adding new records if at the end-of-file. The current line pointer is advanced to the line following the last line written at the end of the operation. Note that writing to a record in the middle of a V-format file does not result in truncation of the file from that point, as it would in the CMS file system.

## CMS Interface for Display Terminals

CMS has an interface that allows it to display large amounts of data in a very rapid fashion. This interface for 3270 display terminals (also 3138, 3148, and 3158) is much faster and has less overhead than the normal write because it displays up to 1760 characters in one operation, instead of issuing 22 individual writes of 80 characters each (that is one write per line on a display terminal). Data that is displayed in the screen output area with this interface is not placed in the console spool file.

The DISPW macro allows you to use this display terminal interface. It generates a calling sequence for the CMS display terminal interface module, DMSGIO. DMSGIO creates a channel program and issues a DIAGNOSE instruction (Code X'58') to display the data. DMSGIO is a TEXT file which must be loaded in order to use DISPW. The format of the CMS DISPW macro is:

| | | |
|---|---|---|
| [label] | DISPW | bufad [ ,LINE=n / ,LINE=0 ] [ ,BYTES=bbbb / ,BYTES=1760 ]<br>[ERASE=YES] [CANCEL=YES] |

*where*:

`label` is an optional macro statement label.

`bufad` is the address of a buffer containing the data to be written to the display terminal.

[LINE=n]
[LINE=0]

is the number of the line, 0 to 23, on the display terminal that is to be written. Line number 0 is the default.

[BYTES=bbbb]
[BYTES=1760]

is the number of bytes (0 to 1760) to be written on the display terminal. 1760 bytes is the default.

[ERASE=YES] specifies that the display screen is to be erased before the current data is written. The screen is erased regardless of the line or number of bytes to be displayed. Specifying ERASE=YES causes the screen to go into "MORE" status.

[CANCEL=YES] causes the CANCEL operation to be performed; the output area is erased.

**Note:** It is advisable for the user to save registers before issuing the DISPW macro and to restore them after the macro, because the modules called by DISPW macro do not save the user's registers. The DISPW macro saves and restores register 13.

# Using the DASD Block I/O System Service from CMS

The DASD Block I/O System Service provides a virtual machine with device-independent access to its virtual DASD devices. Programs using the DASD Block I/O System Service bypass the CMS file system, and they read or write directly from CP.

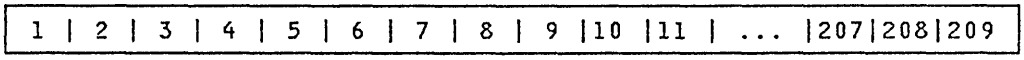
Before using the DASD Block I/O System Service, you should issue the CMS RESERVE command and the CMS DISKID function.

The CMS RESERVE command allocates all available blocks of a 512-, 1K-, 2K-, or 4K-byte block formatted minidisk to a unique CMS file. The file created has the following format:

- filename, filetype, and filemode letter the user specified
- filemode number 6, if the filemode number was not specified in the command
- logical record length equal to the CMS disk block size
- fixed (F) record format
- the number of records is the total number of blocks available on the disk minus the number of blocks used by CMS. The number of blocks used by CMS is referred to as the offset. This CMS overhead varies with the size of the mini-disk. The data blocks physically follow the blocks used by CMS.

The file created can be read or written via the DASD Block I/O System Service or the CMS file system. Because a CMS file structure has been created on the disk, the file may be accessed using the CMS file system. Let's consider the following example:

Suppose you have a 3330 device with one cylinder formatted with 1024-byte block size. There will be 209 blocks available. After you issue the RESERVE command, the file created has the following format:




*where*:

| Physical block number(s) | Description |
|---|---|
| 1 and 2 | Contain the IPL records |
| 3 | Contains the volume label |
| 4 or 5 | Contain the CMS directory file |
| 6 | Contains the allocation map |
| 7 | Contains the alternate allocation map |
| 8 | CMS level 1 pointer block |
| 9 through 209 | Data blocks |

Physical blocks 1 through 8 are the blocks used by the CMS file system. Physical blocks 9 through 209 are the data blocks. Next, issue the following format of the FILEDEF command:

```
FILEDEF ddname DISK vaddr
```

This command associates the name of the virtual minidisk referred to in your program, "ddname", to the virtual address of the minidisk, "vaddr". The "ddname" is input to the DISKID function.

The DISKID function obtains the necessary information on the physical organization of the RESERVEd minidisk that will be used by the DASD Block I/O Service. The DISKID function obtains the virtual address, the block size, and the offset of this minidisk. (In the above example, the block size is 1024 bytes and the offset is 8.)

Before using the DASD Block I/O System Service, you must initialize your virtual machine for IUCV communications. IUCV enables a program running in your virtual machine to communicate with DASD Block I/O. Use the IUCV DECLARE BUFFER function or the CMS HNDIUCV SET macro to initialize your virtual machine for IUCV communications. You should use the CMS IUCV support since this support allows other programs running in the virtual machine to use IUCV.

To establish a path between your virtual machine and the DASD Block I/O System Service, your program must issue either the IUCV CONNECT function or the CMSIUCV CONNECT macro. The USERID parameter on the IUCV CONNECT macro must be "*BLOCKIO" and the PRMDATA parameter must be "YES". PRMDATA=YES indicates, in this case, that the your program will receive messages in its parameter list. Information about the minidisk returned by the DISKID function - the virtual address, blocksize, and offset - must be moved into the IPUSER field of the IUCV CONNECT parameter list.

If all the parameters required by DASD Block I/O are valid, DASD Block I/O issues the IUCV ACCEPT function with the PRMDATA=YES parameter specified. In this case, PRMDATA=YES indicates that the DASD Block I/O System Service will receive messages in its parameter list. If invalid parameters are passed from the CONNECT to DASD Block I/O, DASD Block I/O issues an IUCV SEVER on the path.

If DASD Block I/O issued an IUCV ACCEPT, your virtual machine receives an IUCV Connection Complete external interrupt. The IPUSER field of the Connection Complete external interrupt buffer contains the starting and ending block numbers allowable on the DASD Block I/O requests, and contains flags describing the status of the virtual device. The starting block number (START BLOCK) is 1 minus the offset. The ending block number (END BLOCK) is the total number of available blocks minus the offset.

Since DASD Block I/O bypasses the CMS file system, START BLOCK can contain a negative value (1 - OFFSET) and these blocks can be used by DASD Block I/O. In the example above, START BLOCK would be -7 (1 - 8) and END BLOCK would be 201 (209 - 8). The range -7 to 201 equals 209 - the number of available blocks on the device. Data block 1 is actually physical block 9, data block 2 is actually physical block 10, ..., and the last block (data block 201) is actually physical block 209. Programs using DASD Block I/O should only write data blocks; therefore, a block number less than 1 should never be written. This would destroy a block that was used to implement the CMS file structure.

You can now start sending I/O requests to DASD Block I/O by issuing the IUCV SEND function. You must specify the block number, virtual buffer address, and type of request desired in the IUCV SEND parameter list. Blocks are read or written randomly as requested.

If no error occurred in the IUCV SEND, DASD Block I/O issues an IUCV REPLY to return the results of the I/O requests. If an error occurred in the IUCV SEND, DASD Block I/O issues an IUCV SEVER.

When you want to terminate communications with the DASD Block I/O System Service, issue the IUCV SEVER function, CMS CMSIUCV SEVER macro, IUCV RETRIEVE BUFFER function, or CMS HNDIUCV CLR macro.

(For a further description of the DASD Block I/O System Service, see "DASD Block I/O System Service".)

# CMS IUCV Support

The Inter-User Communications Vehicle (IUCV) is a communications facility. It enables a program running in a virtual machine to communicate with other virtual machines, with a CP system service, and with itself. (See "Inter-User Communication Vehicle" in "Part 1: Control Program (CP)".) CMS support of IUCV makes it easier for multiple programs, operating within one virtual machine, to use IUCV functions.

You can invoke IUCV functions via the CMS macros, HNDIUCV and CMSIUCV. These macros enable you to:

- Initialize and terminate a program's IUCV environment
- Begin or terminate communications with another virtual machine or with CP
- Specify specific exits for IUCV external interrupts.

## HNDIUCV Macro

Use the HNDIUCV macro to identify an IUCV program to CMS. HNDIUCV initializes or terminates the virtual machines IUCV communications. No CMS IUCV function is permitted by a particular program unless the program has first issued the HNDIUCV macro and identified itself to CMS.

The four formats of the HNDIUCV macro are:

- standard,
- MF=L
- MF=(L,addr[,label])
- MF=(E,addr).

### Standard Format

The standard format of the HNDIUCV macro is:

| | | |
|---|---|---|
| [label] | HNDIUCV | { SET,NAME=addr,EXIT=addr[,UWORD=addr][,ERROR=addr]<br>REP,NAME=addr[,ERROR=addr] { ,EXIT=addr / ,UWORD=addr / ,EXIT=addr,UWORD=addr }<br>CLR,NAME=addr[,ERROR=addr] } |

*where:*

`addr` is an assembler program label or an address stored in a general register. If a register is used, it must be enclosed in parentheses. Also, the register must contain a non-zero value. A zero value is treated as though the parameter was not specified, and any defaults are used. If the parameter is required by the macro function, a non-zero return code is generated.

`label` is an assembler program label.

`SET` identifies the program to CMS. It must be issued before doing any CMS IUCV communications. Upon error free completion, register 0 contains the maximum number of possible connections for the virtual machine.

`CLR` removes the program from the list of active CMS IUCV programs. This function should be issued when the program no longer wishes to do any more CMS IUCV communications. Any paths associated with this program are SEVERed when this function is requested (the IPUSER field of the IUCV SEVER parameter list is set to binary ones to indicate the SEVER was done by CMS).

`REP` replaces the currently defined exit address and/or UWORD field for a specified program. Only the parameters specified are replaced.

`NAME=`

`label` is an assembler program label that is the address of an 8 character symbolic name.

`(Rn)` is a general register. Its value is the address of an 8 character symbolic name.

This symbolic name is used as the CMS IUCV program's identity. When this program issues the CMSIUCV macro to perform an IUCV function, the NAME parameter specified on the CMSIUCV macro must be the same as the one specified here. This parameter is required to execute the HNDIUCV function.

`EXIT=`

`label` is an assembler program label that is the address of the exit routine.

`(Rn)` is a general register. Its value is the address of the exit routine.

The exit routine receives control whenever an IUCV external interrupt of the type "PENDING CONNECT" occurs for this program. In order for this exit to be activated, the connecting virtual machine must specify the same symbolic name in the first 8 bytes of the IPUSER field of its CONNECT parameter list as the NAME parameter here. This exit address is the default address associated with any path owned by this program. If an IUCV external interrupt occurs on a path where no specific exit has been established (a pending connect external interrupt has previously occurred on this path and no CMSIUCV ACCEPT has been issued yet, or the EXIT parameter was not specified on the CMSIUCV

CONNECT or ACCEPT that established the path), this address receives control. This parameter must be specified on the SET function, but it is optional on the REP function.

UWORD=

`label` is an assembler program label that is the address stored as the UWORD.

`(Rn)` is a general register. Its contents are stored as the UWORD.

UWORD is an optional fullword that can be specified by the invoker for any purpose desired. When the exit routine receives control, register 0 contains either an address if a label was used or the value of the register if a register was used. If this parameter is not specified, the UWORD is set to zero. (If the UWORD value is not specified when a CMSIUCV ACCEPT or CONNECT is issued, it defaults to the UWORD value specified on this HNDIUCV macro.)

ERROR=

`label` is an assembler program label that is the address of the error routine.

`(Rn)` is a general register. Its value is the address of the error routine.

The error routine receives control if an error is found. If this parameter is not specified and an error occurs, control returns to the next sequential instruction in the calling program.

**MF=L Format**

When MF=L is coded, the format of the HNDIUCV macro is:

| | | |
|---|---|---|
| `[label]` | `HNDIUCV MF=L` | `[,NAME=label][,EXIT=label][,UWORD=label]`<br>`,SET[,NAME=label][,EXIT=label][,UWORD=label]`<br>`,REP[,NAME=label][,EXIT=label][,UWORD=label]`<br>`,CLR[,NAME=label]` |

All parameters have the same meaning as the standard format with the following difference:

`MF=L` indicates that the parameter list is created in-line. No executable code is generated. Register notation cannot be used for macro parameter addresses.

**Note:** When using the MF= parameter, all other parameters are optional. When the function is executed, however, a valid combination of parameters must have been specified by the LIST and EXECUTE formats of the macro.

**MF=(L,addr[,label]) Format**

When MF=(L,addr[,label]) is coded, the format of the HNDIUCV macro is:

| | | |
|---|---|---|
| [label] | HNDIUCV MF=(L,addr[label]) | [,NAME=addr][,EXIT=addr][,UWORD=addr]<br>,SET[,NAME=addr][,EXIT=addr][,UWORD=addr]<br>,REP[,NAME=addr][,EXIT=addr][,UWORD=addr]<br>,CLR[,NAME=addr] |

All parameters have the same meaning as the standard format with the following difference:

MF=(L,addr[,label])
: indicates that the parameter list is created in the area specified by "addr". The address may represent an area within your program or an area of free storage obtained by a system service. You can determine the size of the parameter list by coding the "label" operand. The macro expansion equates "label" to the size of the parameter list. This format of the macro produces executable code to move the data into the parameter list specified by "addr". However, it does not generate instructions to invoke the function. If this version of the LIST format is used, it must be executed before any related invocation of the EXECUTE format.

**Note:** When using the MF= parameter, all other parameters are optional. When the function is executed, however, a valid combination of parameters must have been specified by the LIST and EXECUTE formats of the macro.

### MF=(E,addr) Format

When MF=(E,addr) is coded, the format of the HNDIUCV macro is:

| | | |
|---|---|---|
| [label] | HNDIUCV MF=(E,addr) | [,NAME=addr][,EXIT=addr][,UWORD=addr]<br>[,ERROR=addr]<br>,SET[,NAME=addr][,EXIT=addr][,UWORD=addr]<br>[,ERROR=addr<br>,REP[,NAME=addr][,EXIT=addr][,UWORD=addr]<br>[,ERROR=addr]<br>,CLR[,NAME=addr][,ERROR=addr] |

All parameters have the same meaning as the standard format with the following difference:

MF=(E,addr)
: indicates that instructions are generated to execute the HNDIUCV function. "addr" represents the location of the parameter list. Information in the parameter list may be changed by specifying the appropriate operands on the macro.

**Note:** When using the MF= parameter, all other parameters are optional. When the function is executed, however, a valid combination of parameters must have been specified by the LIST and EXECUTE formats of the macro.

### Error Conditions

If an error occurs, register 15 contains one of the following return codes:

| Code | Meaning |
|---|---|
| 4 | A program with this name has previously issued a HNDIUCV SET (SET) |
| 8 | No HNDIUCV SET has been issued for this program (REP,CLR) |
| 16 | The NAME parameter was not specified or its address is equal to zero (SET,REP,CLR) |
| 20 | The EXIT parameter was not specified or its address is equal to zero (SET) |
| 32 | An IUCV DECLARE BUFFER has already been issued by a non-CMS IUCV program. CMS IUCV support cannot be initialized (SET) |
| 36 | Errors were encountered reading the directory for the virtual machine during CMS IUCV initialization (SET) |
| 40 | Unrecognized function |
| 2xx | An error was encountered in getting CMS free storage. "xx" = the return code from DMSFREE. (SET) |
| 1xxx | While trying to SEVER all of the program's paths, an IUCV SEVER error occurred. "xxx" is the IPRCODE field that was returned by IUCV to aid in diagnosing the error. (CLR) |

## CMSIUCV Macro

Use the CMSIUCV macro to begin or terminate IUCV communications with another IUCV program or with CP.

The four formats of the CMSIUCV macro are:

- standard,
- MF=L
- MF=(L,addr[,label])
- MF=(E,addr).

### Standard Format

The standard format of the CMSIUCV macro is:

| | | |
|---|---|---|
| [label] | CMSIUCV | { CONNECT,NAME=addr,PRMLIST=addr[,EXIT=addr] [,UWORD=addr][,ERROR=addr] ACCEPT,NAME=addr,PRMLIST=addr[,EXIT=addr [,UWORD=addr] [,ERROR=addr] SEVER,NAME=addr,PRMLIST=addr[,ERROR=addr [CODE=ALLONE] } |

*where:*

addr is an assembler program label or an address stored in a general register. If a register is used, it must be enclosed in parentheses. Also, the register must contain a non-zero value. A zero value is treated as though the parameter was not specified, and defaults are used. If the parameter was required by the macro function, a non-zero return code is generated.

label is an assembler program label.

CONNECT requests CMS to perform an IUCV CONNECT. A CONNECT parameter list must be set up by the program and passed to CMS.

ACCEPT requests CMS to perform an IUCV ACCEPT. An ACCEPT parameter list must be set up by the program and passed to CMS.

SEVER requests CMS to perform an IUCV SEVER. A SEVER parameter list must be set up by the program and passed to CMS. Any EXIT established for the path being SEVERed is terminated. A SEVER with the IPALL bit turned on, which would cause a SEVER of all paths for the virtual machine, is not permitted.

EXIT=

label is an assembler program label that is the address of the exit routine.

(Rn) is a general register. Its value is the address of the exit routine.

The exit routine receives control whenever an IUCV external interrupt occurs on this IUCV path. If this parameter is not specified, the exit address defaults to the address specified in the HNDIUCV macro for this program. Any time an IUCV external interrupt occurs for the specific IUCV path, the address is given control.

UWORD=

label is an assembler program label that is the address that is stored as the UWORD.

(Rn) is a general register. Its contents are stored as the UWORD.

UWORD is an optional fullword that can be specified by the invoker for any purpose desired. When the exit routine receives control, register 0 contains the value of the UWORD associated with the path on which the IUCV external interrupt occurred. Register 0 contains the address if a label was used, or the value of the register if a register was used. If this parameter is not specified, the UWORD value defaults to the value specified on the HNDIUCV macro for this program.

PRMLIST=

label is an assembler program label. It is the address of the program's IUCV PRMLIST.

(Rn) is a general register. Its value is the address of the program's IUCV PRMLIST.

This address is the block of storage that contains the IUCV parameter list for the IUCV function desired. This parameter list must be previously prepared by the program. It is suggested that the program use the LIST form of the IUCV macro to prepare the parameter list. By using this form, the program may set up the IUCV parameter list by using KEYWORD parameters on the IUCV macro instead of storing information using the IPARML DSECT. This parameter is required.

`CODE=` is only valid when the SEVER function is requested. If CODE=ALL, all paths owned by the program are SEVERed. If CODE=ONE, only the one path specified via the pathid is SEVERed. If this parameter is not specified, CODE=ONE is used as the default.

`NAME=`

`label` is an assembler program label. It is the address of an 8 character symbolic name.

`(Rn)` is a general register. Its value is the address of an 8 character symbolic name.

This symbolic name identifies the program associated with this path. A program with this name must have previously issued an HNDIUCV macro to identify itself as a CMS IUCV program to CMS. This parameter must be specified.

`ERROR=`

`label` is an assembler program label that is the address of the error routine.

`(Rn)` is a general register. Its value is the address of the error routine.

The error routine receives control if an error is found. If this parameter is not specified and an error occurs, control returns to the next sequential instruction in the calling program.

**MF=L Format**

When MF=L is coded, the format of the CMSIUCV macro is:

| | | |
|---|---|---|
| `[label]` | `CMSIUCV MF=L` | `[,NAME=label][,PRMLIST=label][,EXIT=label]`<br>`[,UWORD=label][,CODE=ALLONE]`<br>`,CONNECT[,NAME=label][,PRMLIST=label][,EXIT=label]`<br>`[,UWORD=label]`<br>`,ACCEPT[,NAME=label][,PRMLIST=label][,EXIT=label]`<br>`[,UWORD=label]`<br>`,SEVER[,NAME=label][,PRMLIST=label][,CODE=ALLONE]` |

All parameters have the same meaning as the standard format with the following difference:

MF=L indicates that the parameter list is created in-line. No executable code is generated. Register notation cannot be used for macro parameter addresses.

**Note:** When using the MF= parameter, all other parameters are optional. When the function is executed, however, a valid combination of parameters must have been specified by the LIST and EXECUTE formats of the macro.

**MF=(L,addr[,label]) Format**

When MF=(L,addr[,label]) is coded, the format of the CMSIUCV macro is:

| | | |
|---|---|---|
| [label] | CMSIUCV MF=(L,addr[,label]) | [,NAME=addr][,PRMLIST=addr][,EXIT=addr]<br>[,UWORD=addr][,CODE=ALLONE]<br>,CONNECT[,NAME=addr][,PRMLIST=addr]<br>[,EXIT=addr][,UWORD=addr]<br>,ACCEPT[,NAME=addr][,PRMLIST=addr]<br>[,EXIT=addr][,UWORD=addr]<br>,SEVER[,NAME=addr][,PRMLIST=addr]<br>[CODE=ALLONE] |

All parameters have the same meaning as the standard format with the following difference:

MF=(L,addr ,label )
indicates that the parameter list is created in the area specified by "addr". The address may represent an area within your program or an area of free storage obtained by a system service. You can determine the size of the parameter list coding the "label" operand. The macro expansion equates "label" to the size of the parameter list. This format of the macro produces executable code to move the data in the parameter list specified by "addr". However, it does not generate instructions to invoke the function. If this version of the LIST format is used, it must be executed before any related invocation of the EXECUTE format.

**Note:** When using the MF= parameter, all other parameters are optional. When the function is executed, however, a valid combination of parameters must have been specified by the LIST and EXECUTE formats of the macro.

**MF=(E,addr) Format**

When MF=(E,addr) is coded, the format of the CMSIUCV macro is:

| | | |
|---|---|---|
| [label] | CMSIUCV MF=(E,addr) | [,NAME=addr][,PRMLIST=addr][,EXIT=addr]<br>[,UWORD=addr][,ERROR=addr][,CODE=ALLONE]<br>,CONNECT[,NAME=addr][,PRMLIST=addr][,EXIT=addr]<br>[,UWORD=addr][,ERROR=addr]<br>,ACCEPT[,NAME=addr][,PRMLIST=addr][,EXIT=addr]<br>[,UWORD=addr][,ERROR=addr]<br>,SEVER[,NAME=addr][,PRMLIST=addr][CODE=ALLONE]<br>[,ERROR=addr] |

All parameters have the same meaning as the standard format with the following difference:

MF=(E,addr)
: indicates that instructions are generated to execute the CMSIUCV function. "addr" represents the location of the parameter list. Information in the parameter list may be changed by specifying the appropriate operands on the macro.

**Note:** When using the MF= parameter, all other parameters are optional. When the function is executed, however, a valid combination of parameters must have been specified by the LIST and EXECUTE formats of the macro.

**Usage Notes:**

1. To insure that no program tries to SEVER a path that another program established, each individual IUCV path has a NAME associated with it. When a program requests a CONNECT or ACCEPT function, the NAME specified becomes the owner of this path. If the program requests a SEVER or an ACCEPT for a specific path and the NAME specified does not correspond with the owner of that path, the SEVER or ACCEPT is not permitted.

2. The HNDIUCV macro must be issued to identify the program to CMS before issuing the CMSIUCV macro.

3. If the program requests a SEVER function with CODE=ALL, all IUCV paths owned by that program are SEVERed. The IPUSER field of the IUCV SEVER PRMLIST is set to binary ones.

4. The IUCV communication facility generates exceptions for some error conditions. If IUCV generates an operation, specification, or addressing exception while a HNDIUCV or CMSIUCV macro is executing, control does not directly return to the next sequential instruction. Instead, a program check is generated.

5. The HNDIUCV REP function will only replace the general exit address and/or UWORD set up by your program via the HNDIUCV SET function. If your program had previously issued any CMSIUCV CONNECTs and had the EXIT address or UWORD default to the HNDIUCV SET's EXIT and UWORD, the HNDIUCV REP function does not replace the path specific EXIT or UWORD set up via the CMSIUCV function. The EXIT and UWORD remain as established when the CMSIUCV function was issued.

**Error Conditions:**

If an error occurs, register 15 contains one of the following return codes:

| Code | Meaning |
|---|---|
| 8 | No HNDIUCV SET has been issued for this program (CONNECT,ACCEPT,SEVER) |
| 12 | The program doesn't own the path (ACCEPT,SEVER) |
| 16 | The NAME parameter was not specified or its address is equal to zero (CONNECT,ACCEPT,SEVER) |
| 24 | The PRMLIST parameter was not specified or its address is equal to zero (CONNECT,ACCEPT,SEVER) |

| Code | Meaning |
|---|---|
| 28 | An IUCV SEVER with the IPALL bit on is not allowed (SEVER) |
| 40 | Unrecognized function |
| 1xxx | Indicates that an IUCV error occurred. "xxx" is the IPRCODE field that was returned by IUCV to aid in diagnosing the error. (CONNECT,ACCEPT,SEVER) |

## Exits

When the program's IUCV external interruption routine is given control, all interruptions are disabled. The exit routine is responsible for providing proper entry and exit linkage for its IUCV external interruption handling routine. The exit routine has the following requirements:

- The routine should not enable itself for any type of interrupts.
- The routine should not perform any I/O operations, since all interruptions are disabled.
- The routine must return control to the address in register 14.

When the routine receives control, the significant registers contain:

| Register | Contents |
|---|---|
| 0 | UWORD Field |
| 1 | Points to a SAVEAREA in the format: |

| Label | Displacement Dec | Hex |
|---|---|---|
| GRS | 0 | 0 |
| FRS | 64 | 40 |
| PSW | 96 | 60 |
| UAREA | 104 | 68 |
| END | 176 | 80 |

| | |
|---|---|
| 2 | Address of the IUCV External Interrupt Buffer |
| 13 | Points to the savearea at label UAREA for use by the exit routine |
| 14 | Return address |
| 15 | Entry point address |

## Usage Notes

1. If the CMS IUCV support is active, the external interrupt handler recognizes two error conditions.
   - An IUCV pending-connect external interrupt occurs and the first eight bytes of the IPUSER field does not match any currently active CMS IUCV program's identity.

- Any other type of IUCV external interrupts occurs and the path that it occurs on is not owned by any active CMS IUCV programs in the virtual machine.

In either condition, CMS issues an IUCV SEVER for the path in error. The 16 bytes in the IPUSER field contain binary ones (X'F').

2. If a CMS abend occurs, the CMS IUCV environment is terminated. An IUCV RETRIEVE BUFFER is issued, and any exits set up by the CMSIUCV or HNDIUCV macros are cancelled.

3. CMS IUCV clean up does not occur at end-of-command processing.

4. A program must be ready to handle any incoming external interrupts as soon as a HNDIUCV or CMSIUCV macro has finished execution. A program may even be interrupted before the next sequential instruction after the macro in the program is executed.

## Using CMS IUCV to Communicate Between Two Virtual Machines

Figure 41 on page 366 illustrates the sequence of macro instructions issued when a virtual machine communicates with another virtual machine using CMS IUCV.

The functions performed by these instructions include:

- initializing IUCV communications
- connecting to another virtual machine
- sending and receiving messages
- replying to and waiting for messages
- severing connections with the other virtual machine
- terminating IUCV communications.

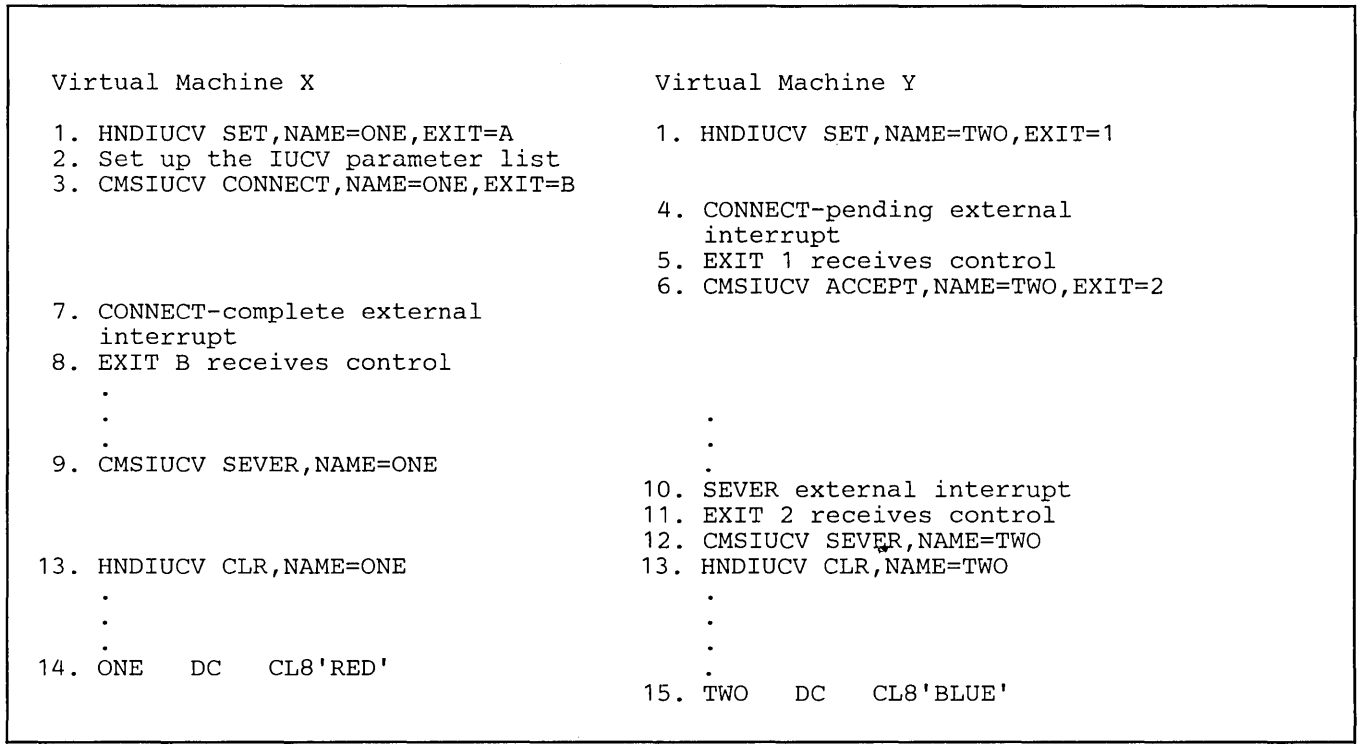



**Figure 41. Sequence of Instructions in Virtual Machine to Virtual Machine Communication**

The following list is an explanation of the sequence of instructions used above.

1. A program running in virtual machine X wishes to communicate with a program running in virtual machine Y. Each program must independently issue the HNDIUCV macro to begin IUCV communications. By issuing HNDIUCV SET, CMS invokes the IUCV DECLARE BUFFER function. The EXIT parameter establishes a general exit to handle IUCV CONNECT PENDING external interrupts.

2. Before issuing a CMSIUCV CONNECT, an IUCV CONNECT parameter list must be set up by the program. The IPVMID field of the IUCV parameter list contains the userid of the virtual machine you are connecting to (virtual machine Y). The first 8 bytes of the IPUSER field of the IUCV parameter list contains the eight-character identifying name of the program that issued a HNDIUCV SET in virtual machine Y. This name must match the name specified on the HNDIUCV macro issued by the program in virtual machine Y. In this example, the first eight bytes of the IPUSER field equals BLUE.

3. The program in virtual machine X issues a CMSIUCV CONNECT to initiate a communication link with virtual machine Y. By issuing CMSIUCV CONNECT, CMS invokes the IUCV CONNECT function. This associates the exit address, "B", with the IUCV pathid.

4. Virtual machine Y receives a CONNECT-pending external interrupt as a result of the CMSIUCV CONNECT issued by the program in virtual machine X.

5. "EXIT 1" receives control as a result of the external interrupt. ("EXIT 1" receives control because it was specified on the EXIT parameter of the HNDIUCV macro.)

6. To complete the connection, the program in virtual machine Y issues a CMSIUCV ACCEPT. By issuing CMSIUCV ACCEPT, CMS invokes the IUCV ACCEPT function. This completes the IUCV communication link with virtual machine X. The CMSIUCV ACCEPT also associates the exit address, "2", with the pathid.

7. Virtual machine X receives a connection-complete external interrupt as a result of the CMSIUCV ACCEPT issued by the program in virtual machine Y.

8. "EXIT B" receives control as a result of the external interrupt. ("EXIT B" receives control because it is specified on the EXIT parameter of the CMSIUCV macro.)

9. Virtual machine X completed its communications with virtual machine Y and terminates the IUCV communication link. The program in virtual machine X issues an CMSIUCV SEVER to terminate this link. By issuing CMSIUCV SEVER, CMS invokes the IUCV SEVER function and clears the exit associated with the communication link.

10. Virtual machine Y receives a SEVER external interrupt as a result of the CMSIUCV SEVER issued by virtual machine X.

11. "EXIT 2" receives control as a result of the external interrupt. ("EXIT 2" receives control because it was specified on the EXIT parameter of the CMSIUCV macro.)

12. The program in virtual machine Y issues a CMSIUCV SEVER to terminate the communication link. By issuing CMSIUCV SEVER, CMS invokes the IUCV SEVER function and clears the exit associated with the communication link.

13. After all communications are complete and all communication paths have been SEVERed, the program in virtual machine X and the program in virtual machine Y independently issue HNDIUCV CLR. HNDIUCV CLR terminates IUCV communications and clears the general exit for IUCV PENDING CONNECTs. CMS invokes the IUCV RETRIEVE BUFFER function if there are no other programs in the virtual machine using IUCV.

14. This is the label specified in the NAME parameter. This location contains the identifying name of the program in virtual machine X. The name of this program is RED.

15. This is the label specified in the NAME parameter. This location contains the identifying name of the program in virtual machine Y. The name of this program is BLUE.

## Guidelines and Limitations of the CMS IUCV Support

Some of the existing IUCV functions affect the IUCV environment of the entire virtual machine. Since CMS cannot intercept any IUCV functions directly issued by a program, any program using the CMS IUCV support has certain limitations on its use of IUCV functions. The program must not issue any IUCV function that alters the virtual machine's IUCV environment.

The following is a list of IUCV functions. The list describes their relationship to the CMS IUCV support and some guidelines for their usage. If any functions listed

as "Should not be used" are indeed used, other programs using CMS IUCV functions in the virtual machine may be affected. For information on coding the following functions, see the "Inter-User Communications Vehicle" earlier in this manual.

ACCEPT
: is invoked by a program via the CMSIUCV macro. It should not be issued directly by a program.

CONNECT
: is invoked by a program via the CMSIUCV macro. It should not be issued directly by a program.

DECLARE BUFFER
: is used by HNDIUCV to initialize the virtual machine's IUCV environment. It should not be issued directly by a program.

DESCRIBE
: should not be used because this function clears the pending-message external interruption for the described message. This interrupt may not belong to the issuer of the DESCRIBE function. Thus, other programs running in the same virtual machine may be affected since the message is lost and never reflected to the true target.

PURGE
: is issued directly by a program.

QUERY
: is used by HNDIUCV to determine the size of the external interrupt buffer and the maximum number of connections for this virtual machine. It may be issued directly by an application program.

QUIESCE
: is issued directly by a program to quiesce a specific path. However, the issuer must be careful that the IPALL bit is not turned on in the IPFLAGS1 byte of the parameter list. This would quiesce all paths in the virtual machine.

RECEIVE
: is issued directly by the application program. However, the issuer must be careful that a specific message id or path id is specified in the IUCV parameter list. If it is not, IUCV RECEIVEs the first message that has not yet been partially received for the entire virtual machine. This message may not belong to the program that issued the IUCV RECEIVE.

REJECT
: is issued directly by a program.

REPLY
: is issued directly by a program.

RESUME
: is issued directly by a program in order to resume a specific path. However, the issuer must be careful that the IPALL bit is not turned on in the IPFLAGS1 byte of the parameter list. This would resume all paths in the virtual machine.

RETRIEVE BUFFER
: is used by HNDIUCV and CMS ABEND processing to terminate the virtual machine's IUCV environment. It should not be issued directly by a program.

SEND
: is issued directly by a program.

SETMASK
: should not be used because this function disables certain IUCV external interrupts for the entire virtual machine. Thus, other programs running in the same virtual machine may be affected.

SETCMASK
: should not be used because this function disables certain IUCV external interrupts for the entire virtual machine. Thus, other programs running in the same virtual machine may be affected.

SEVER
: is invoked by a program via the CMSIUCV macro. This IUCV function may be invoked to SEVER all existing paths for the CMS IUCV program that has issued the HNDIUCV CLR macro. This IUCV function should not be issued directly by a program.

TEST COMPLETION
: is issued directly by a program. However, the issuer must be careful that a specified message id or path id is specified in the IUCV parameter list. If it is not, IUCV completes the first message on the REPLY queue for the entire virtual machine. This message may not belong to the application that issued the TEST COMPLETION.

TEST MESSAGE
: should not be used because this function places the entire virtual machine in a wait state if no incoming messages or replies are pending. Thus, other programs running in the same virtual machine may be affected.

# OS Macro Simulation Under CMS

When a language processor or a user-written program is executing in the CMS environment and using OS-type functions, it is not executing OS code. Instead, CMS provides routines that simulate the OS functions required to support OS language processors and their generated object code.

CMS functionally simulates the OS macros in a way that presents equivalent results to programs executing under CMS. The OS macros are supported only to the extent stated in the publications for the supported language processors, and then only to the extent necessary to successfully satisfy the specific requirement of the supervisory function.

The restrictions for COBOL and PL/I program execution, listed in "Executing a Program that Uses OS Macros" in the *VM/SP Planning Guide and Reference*, exist because of the limited CMS simulation of the OS macros.

Figure 42 on page 371 shows the OS macro functions that are partially or completely simulated, as defined by SVC number.

## OS Data Management Simulation

The disk format and data base organization of CMS are different from those of OS. A CMS file produced by an OS program running under CMS and written on a CMS disk, has a different format from that of an OS data set produced by the same OS program running under OS and written on an OS disk. The data is exactly the same, but its format is different. (An OS disk is one that has been formatted by an OS program, such as the Device Support Facility.)

### *Handling Files that Reside on CMS Disks*

CMS can read, write, or update any OS data that resides on a CMS disk. By simulating OS macros, CMS simulates the following access methods so that OS data organized by these access methods can reside on CMS disks:

| | |
|---|---|
| direct | identifying a record by a key or by its relative position within the data set. |
| partitioned | seeking a named member within the data set. |
| sequential | accessing a record in a sequence in relation to preceding or following items in the data set. |

Refer to Figure 42 on page 371 and the "Simulation Notes," then read "Access Method Support" to see how CMS handles these access methods.

Since CMS does not simulate the indexed sequential access method (ISAM), no OS program that uses ISAM can execute under CMS. Therefore, no program can write an indexed sequential data set on a CMS disk.

### *Handling Files that Reside on OS or DOS Disks*

By simulating OS macros, CMS can read, but not write or update, OS sequential and partitioned data sets that reside on OS disks. Using the same simulated OS

macros, CMS can read VSE sequential files that reside on DOS disks. The OS macros handle the VSE data as if it were OS data. Thus, a VSE sequential file can be used as input to an OS program running under CMS.

However, an OS sequential or partitioned data set that resides on an OS disk can be written or updated only by an OS program running in a real OS machine.

CMS can execute programs that read and write VSAM files from OS programs written in the VS BASIC, COBOL, PL/I, VS/APL, and VS FORTRAN programming languages. CMS also supports VSAM for use with DOS/VS SORT/MERGE. This CMS support is based on the VSE/VSAM program product and, therefore, the OS user is limited to those VSAM functions that are available under VSE/VSAM.

| Macro Name | SVC Number | Function |
|---|---|---|
| XDAP[1] | 00 | Read or write direct access volumes |
| WAIT | 01 | Wait for an I/O completion |
| POST | 02 | Post the I/O completion |
| EXIT/RETURN | 03 | Return from a called phase |
| GETMAIN | 04 | Conditionally acquire user storage |
| FREEMAIN | 05 | Release user-acquired storage |
| GETPOOL | - | Simulate as SVC 10 |
| FREEPOOL | - | Simulate as SVC 10 |
| LINK | 06 | Link control to another phase |
| XCTL | 07 | Delete, then link control to another load phase |
| LOAD | 08 | Read a phase into storage |
| DELETE | 09 | Delete a loaded phase |
| GETMAIN/ FREEMAIN | 10 | Manipulate user free storage |
| TIME[1] | 11 | Get the time of day |
| ABEND | 13 | Terminate processing |
| SPIE[1] | 14 | Allow processing program to handle program interrupts. |
| RESTORE[1] | 17 | Effective NOP |
| BLDL/FIND[1] | 18 | Manipulate simulated partitioned data files |
| OPEN | 19 | Activate a data file |
| CLOSE | 20 | Deactivate a data file |
| STOW[1] | 21 | Manipulate partitioned directories |
| OPENJ | 22 | Activate a data file |

**Figure 42 (Part 1 of 2). Simulated OS Supervisor Calls**

| Macro Name | SVC Number | Function |
|---|---|---|
| TCLOSE | 23 | Temporarily deactivate a data file |
| DEVTYPE[1] | 24 | Get device-type physical characteristics |
| TRKBAL | 25 | NOP |
| FEOV | 31 | Set forced EOV error code |
| WTO/WTOR[1] | 35 | Communicate with the terminal |
| EXTRACT[1] | 40 | Effective NOP |
| IDENTIFY[1] | 41 | Add entry to loader table |
| ATTACH[1] | 42 | Effective LINK |
| CHAP[1] | 44 | Effective NOP |
| TTIMER[1] | 46 | Access or cancel timer |
| STIMER[1] | 47 | Set timer |
| DEQ[1] | 48 | Effective NOP |
| SNAP[1] | 51 | Dump specified areas of storage |
| ENQ[1] | 56 | Effective NOP |
| FREEDBUF | 57 | Release a free storage buffer |
| STAE | 60 | Allow processing program to decipher abend conditions |
| DETACH[1] | 62 | Effective NOP |
| CHKPT[1] | 63 | Effective NOP |
| RDJFCB[1] | 64 | Obtain information from FILEDEF command |
| SYNAD[1] | 68 | Handle data set error conditions |
| BSP[1] | 69 | Back up a record on a tape or disk |
| GET/PUT | - | Access system-blocked data |
| READ/WRITE | - | Access system-record data |
| NOTE/POINT | - | Manage data set positioning |
| CHECK | - | Verify READ/WRITE completion |
| TGET/TPUT | 93 | Read or write a terminal line |
| TCLEARQ | 94 | Clear terminal input queue |
| STAX | 96 | Update a queue of CMTAXs |
| PGRLSE[1] | 112 | Release storage contents |

**Figure 42 (Part 2 of 2). Simulated OS Supervisor Calls**

[1]Simulated in the routine DMSSVT. Other simulation routines reside in the nucleus.

## *Simulation Notes*

Because CMS has its own file system and is a single-user system operating in a virtual machine with virtual storage, there are certain restrictions for the simulated OS function in CMS. For example, HIARCHY options and options that are used only by OS multitasking systems are ignored by CMS.

Due to the design of the CMS loader, an XCTL from the explicitly loaded phase, followed by a LINK by succeeding phases, may cause unpredictable results.

Listed below are descriptions of all the OS macro functions that are simulated by CMS as seen by the programmer. Implementation and program results that differ from those given in *OS Data Management Macro Instructions* and *OS Supervisor Services and Macro Instructions* are stated. HIARCHY options and those used only by OS multitasking systems are ignored by CMS. Validity checking is not performed within the simulation routines. The entry point name in LINK, XCTL, and LOAD (SVC 6, 7, 8) must be a member name or alias in a TXTLIB directory unless the COMPSWT is set to on. If the COMPSWT is on, SVC 6, 7, and 8 must specify a module name. This switch is turned on and off by using the COMPSWT macro. See the *VM/SP CMS Command and Macro Reference* for descriptions of all CMS user macros.

| Macro-SVC No. | Differences in Implementation |
|---|---|
| XDAP-SVC0 | The TYPE option must be R or W; the V, I, and K options are not supported. The BLKREF-ADDR must point to an item number acquired by a NOTE macro. Other options associated with V, I, or K are not supported. |
| WAIT-SVC1 | All options of WAIT are supported. The WAIT routine waits for the completion bit to be set in the specified ECBs. |
| POST-SVC2 | All options of POST are supported. POST sets a completion code and a completion bit in the specified ECB. |
| EXIT/RETURN-SVC3 | Post ECB, execute end of task routines, release phase storage, unchain and free latest request block, and restore registers depending upon whether this is an exit or return from a linked or an attached routine. Do not use EXIT/RETURN to exit from an explicitly LOADed phase. If EXIT/RETURN is used for this purpose, CMS issues abend code A0A. |
| GETMAIN-SVC4 | All options of GETMAIN are supported except SP, BNDRY=, and HIARCHY, which are ignored by CMS, and LC and LU, which result in abnormal termination if used. GETMAIN gets blocks of free storage. |
| FREEMAIN-SVC5 | All options of FREEMAIN are supported except SP, which is ignored by CMS, and L, which result in abnormal termination if used. FREEMAIN frees blocks of storage acquired by GETMAIN. |
| LINK-SVC6 | The DCB and HIARCHY options are ignored by CMS. All other options of LINK are supported. LINK loads the specified program into storage (if necessary) and passes control to the specified entry point. |

XCTL-SVC7 The DCB and HIARCHY options are ignored by CMS. All other options of XCTL are supported. XCTL loads the specified program into storage (if necessary) and passes control to the specified entry point.

LOAD-SVC8 The DCB and HIARCHY options are ignored by CMS. All other options of LOAD are supported. LOAD loads the specified program into storage (if necessary) and returns the address of the specified entry point in register zero. If loading a subroutine is required when SVC8 is issued, CMS searches directories for a TXTLIB member containing the entry point or for a TEXT file with a matching filename. An entry name in an unloaded TEXT file will not be found unless the filename matches the entry name. After the subroutine is loaded, CMS attempts to resolve external references within the subroutine, and may return another entry point address. To insure a correct address in register zero, the user should bring such subroutines into storage either by the CMS LOAD/INCLUDE commands or by a VCON in the user program.

GETPOOL/FREEPOOL
All the options of GETPOOL and FREEPOOL are supported. GETPOOL constructs a buffer pool and stores the address of a buffer pool control block in the DCB. FREEPOOL frees a buffer pool constructed by GETPOOL.

DELETE-SVC9 All the options of DELETE are supported. DELETE decreases the use count by one and, if the result is zero, frees the corresponding virtual storage. Code 4 is returned in register 15 if the phase is not found.

GETMAIN/FREEMAIN-SVC10
All the options of GETMAIN and FREEMAIN are supported except SP and HIARCHY, which are ignored by CMS.

TIME-SVC11 CMS supports only the DEC, BIN, TU, and MIC parameters of the TIME macro instruction. However, the time value that CMS returns is only accurate to the nearest second, and is converted to the proper unit.

ABEND-SVC13 The completion code parameter is supported. The DUMP parameter is not. If a STAE request is outstanding, control is given to the proper STAE routine. If a STAE routine is not outstanding, a message indicating that an abend has occurred is printed on the terminal along with the completion code.

SPIE-SVC14 All the options of SPIE are supported. The SPIE routine specifies interruption exit routines and program interruption types that cause the exit routine to receive control.

RESTORE-SVC17
The RESTORE routine in CMS is a NOP. It returns control to the user.

BLDL-SVC18 BLDL is an effective NOP for LINKLIBs and JOBLIBs. For TXTLIBs and MACLIBs, item numbers are filled in the TTR

field of the BLDL list; the K, Z, and user data fields, as described in *OS/VS Data Management Macro Instructions,* are set to zeros. The "alias" bit of the C field is supported, and the remaining bits in the C field are set to zero.

FIND-SVC18
: All the options of FIND are supported. FIND sets the read/write pointer to the item number of the specified member.

STOW-SVC21
: All the options of STOW are supported. The "alias" bit is supported, but the user data field is not stored in the MACLIB directory since CMS MACLIBs do not contain user data fields.

OPEN/OPENJ-SVC19/22
: All the options of OPEN and OPENJ are supported except for the DISP, EXTEND, and RDBACK options, which are ignored. OPEN creates a CMSCB (if necessary), completes the DCB, and merges necessary fields of the DCB and CMSCB.

CLOSE/TCLOSE-SVC20/23
: All the options of CLOSE and TCLOSE are supported except for the DISP option, which is ignored. The DCB is restored to its condition before OPEN. If the device type is disk, the file is closed. If the device type is tape, the REREAD option is treated as a REWIND. For TCLOSE, the REREAD option is REWIND, followed by a forward space file for tapes with standard labels.

DEVTYPE-SVC24
: With the exception of the RPS option, which CMS ignores, CMS accepts all options of the DEVTYPE macro instruction. In supporting this macro instruction, CMS groups all devices of a particular type into the same class. For example, all printers are grouped into the printer class, all tape drives into the tape drive class, and so forth. In response to the DEVTYPE macro instruction, CMS provides the same device characteristics for all devices in a particular class. Thus, all devices in a particular class appear to be the same device type.

  The device type characteristics CMS returns for each class are:

| Class | Device Characteristics |
|---|---|
| Printer | 1403 |
| Virtual reader | 2540 |
| Console | 1052 |
| Tape drive | 2400 (9 track) |
| DASD | 2314 |
| Virtual punch | 2540 |
| DUMMY | 2314 |
| unassigned | 2314 |

FEOV-SVC31
: Control is returned to CMS with an error code of 4 in register 15.

WTO/WTOR-SVC35
: All options of WTO and WTOR are supported except those options concerned with multiple console support. WTO displays

a message at the operator's console. WTOR displays a message at the operator's console, waits for a reply, moves the reply to the specified area, sets a completion bit in the specified ECB, and returns. There is no check made to determine if the operator provides a reply that is too long. The reply length parameter of the WTOR macro instruction specifies the maximum length of the reply. The WTOR macro instruction reads only this amount of data.

EXTRACT-SVC40
: The EXTRACT routine in CMS is essentially a NOP. The user-provided answer area is set to zeros and control is returned to the user with a return code of 4 in register 15.

IDENTIFY-SVC41
: The IDENTIFY routine in CMS adds a RPQUEST block to the load request chain for the requested name and address.

ATTACH-SVC42
: All the options of ATTACH are supported in CMS as in OS PCP. The following options are ignored by CMS: DCB, LPMOD, DPMOD, HIARCHY, GSPV, GSPL, SHSPV, SHSPL, SZERO, PURGE, ASYNCH, and TASKLIB. ATTACH passes control to the routine specified, fills in an ECB completion bit if an ECB is specified, passes control to an exit routine if one is specified, and returns control to the instruction following the ATTACH.

  Since CMS is not a multitasking system, a phase requested by the ATTACH macro must return to CMS.

CHAP-SVC44
: The CHAP routine in CMS is a NOP. It returns control to the user.

TTIMER-SVC46
: All the options of TTIMER are supported.

STIMER-SVC47
: All options of STIMER are supported except for TASK and WAIT. The TASK option is treated as if the REAL option had been specified, and the WAIT option is treated as a NOP; it returns control to the user. The maximum time interval allowed is X'7FFFFF00' timer units (or 15 hours, 32 minutes, and 4 seconds in decimal). If the time interval is greater than the maximum, it is set to the maximum.

  **Note:** If running in the CMSBATCH environment, issuing the STIMER or TTIMER macro affects the CMSBATCH time limit. Depending on the frequency, number, and duration of STIMER and/or TTIMERS issued, the CMSBATCH limit may never expire.

DEQ-SVC48
: The DEQ routine in CMS is a NOP. It returns control to the user.

SNAP-SVC51
: Except for SDATA, PDATA, and DCB, all options of the SNAP macro are processed normally. SDATA and PDATA are ignored. Processing for the DCB option is as follows. The

DBC address specified with SNAP is used to verify that the file associated with the DCB is open. If it is not open, control is returned to the caller with a return code of 4. If the file is open, then storage is dumped (unless the FCB indicates a DUMMY device type). SNAP always dumps output to the printer. The dump contains the PSW, the registers, and the storage specified.

ENQ-SVC56 The ENQ routine in CMS is a NOP. It returns control to the user.

FREEDBUF-SVC57
All the options of FREEDBUF are supported. FREEDBUF returns a buffer to the buffer pool assigned to the specified DCB.

STAE-SVC60 All the options of STAE are supported except for the XCTL option, which is set to XCTL=YES; the PURGE option, which is set to HALT; and the ASYNCH option, which is set to NO. STAE creates, overlays, or cancels a STAE control block as requested. STAE retry is not supported.

DETACH-SVC62
The DETACH routine in CMS is a NOP. It returns control to the user.

CHKPT-SVC63 The CHKPT routine is a NOP. It returns control to the user.

RDJFCB-SVC64 All the options of RDJFCB are supported. RDJFCB causes a Job File Control Block (JFCB) to be read from a CMS Control Block (CMSCB) into real storage for each data control block specified. CMSCBs are created by FILEDEF commands.

Additional information regarding CMS 'OS Simulation' of RDJFCB follows:

- The DCBs specified in the RDJFCB PARAMETER LIST are processed sequentially as they appear in the parameter list.
- On return to the caller, a return code of zero is always placed in register 15. If an abend occurs, control is not returned to the caller.
- Abend 240 occurs if zero is specified as the address of the area into which the JFCB is to be placed.
- Abend 240 occurs if a JFCB EXIT LIST ENTRY (Entry type X'07') is not present in the DCB EXIT LIST for any one of the DCBs specified in the RDJFCB PARAMETER LIST.
- If a DCB is encountered in the parameter list with zero specified as the DCB EXIT LIST ('EXLST') address, the RDJFCB immediately returns with return code zero in reg-

ister 15. Except for this situation, all of the DCBs specified in the RDJFCB PARAMETER LIST are processed, unless an abend occurs.

- For a DCB that is not open, a search is done for the corresponding FILEDEF or DLBL. If one is not found, a test is done to determine if a file exists with a filename of 'FILE', a filetype of the DDNAME from DCB, and a filemode of 'A1'. If such a file does exist, then X'40' is placed in the JFCB at displacement X'57' (FLAG 'JFCOLD IN FIELD 'JFCBIND2'). If such a file does not exist then X'C0' (FLAG 'JFCNEW') will be in field 'JFCBIND2'.

- For a file that is not open, but for which a DLBL has been specified, X'08' is placed in the JFCB at displacement X'63' (FIELD 'JFCDSORG' BYTE 2) to indicate that it is a VSAM file.

SYNADAF-SVC68
: All the options of SYNADAF are supported. SYNADAF analyzes an I/O error and creates an error message in a work buffer.

SYNADRLS-SVC68
: All the options of SYNADRLS are supported. SYNADRLS frees the work area acquired by SYNAD and deletes the work area from the save area chain.

BSP-SVC69
: All the options of BSP are supported. BSP decrements the item pointer by one block.

TGET/TPUT-SVC93
: TGET and TPUT operate as if EDIT and WAIT were coded. TGET reads a terminal line. TPUT writes a terminal line.

TCLEARQ-SVC94
: TCLEARQ in CMS clears the input terminal queue and returns control to the user.

STAX-SVC96
: The only option of STAX that is supported is EXIT ADDRESS. STAX updates a queue of CMTAXEs each of which defines an attention exit level.

PGRLSE-SVC112
: Release all complete pages (4K bytes) associated with the area of storage specified.

NOTE
: All the options of NOTE are supported. NOTE returns the relative position of the last block read or written.

POINT
: All the options of POINT are supported. POINT causes the control program to start processing the next read or write operation at the specified item number. The TTR field in the block address is used as an item number.

CHECK — All the options of CHECK are supported. CHECK tests the I/O operation for errors and exceptional conditions.

DCB — The following fields of a DCB may be specified, relative to the particular access method indicated:

| Operand | BDAM | BPAM | BSAM | QSAM |
|---|---|---|---|---|
| BFALN | F,D | F,D | F,D | F,D |
| BLKSIZE | n(number) | n | n | n |
| BUFCB | a(address) | a | a | a |
| BUFL | n | n | n | n |
| BUFNO | n | n | n | n |
| DDNAME | s(symbol) | s | s | s |
| DSORG | DA | PO | PS | PS |
| EODAD | - | a | a | a |
| EXLST | a | a | a | a |
| KEYLEN[8] | n | - | n | - |
| LIMCT | n | - | - | - |
| LRECL | - | n | n | n |
| MACRF | R,W | R,W | R,W,P | G,P,L,M |
| OPTCD | A,E,F,R | - | J | J |
| RECFM | F,V,U | F,V,U | F,V,B,S,A,M,U | F,V,B,U,A,M,S |
| SYNAD | a | a | a | a |
| NCP | - | n | n | - |

## *Access Method Support*

The manipulation of data is governed by an access method. To facilitate the execution of OS Code under CMS, the processing program must see data as OS would present it. For instance, when the processors expect an access method to acquire input source cards sequentially, CMS invokes specially written routines that simulate the OS sequential access method and pass data to the processors in the format that the OS access methods would have produced. Therefore, data appears in storage as if it had been manipulated using an OS access method. For example, block descriptor words (BDW), buffer pool management, and variable records are updated in storage as if an OS access method had processed the data. The actual writing to and reading from the I/O device is handled by CMS file management. Note that the character string X'61FFFF61' is interpreted by CMS as an end of file indicator.

The essential work of the volume table of contents (VTOC) and the data set control block (DSCB) is done in CMS by a master file directory (MFD) which updates the disk contents, and a file status table (FST) (one for each data file). All disks are formatted in physical blocks of 800 bytes.

CMS continues to update the OS format, within its own format, on the auxiliary device, for files whose filemode number is 4. That is, the block and record descriptor words (BDW and RDW) are written along with the data. If a data set consists of blocked records, the data is written to, and read from, the I/O device in physical blocks, rather than logical records. CMS also simulates the specific methods of manipulating data sets.

---

8 If an input data set is not a BDAM data set, zero is the only value that should be specified for KEYLEN. This applies to the user exit lists as well as to the DCB macro instruction.

When the OPEN macro instruction is executed, the CMS simulation of the OS OPEN routine initializes the Data Control Block (DCB). The DCB fields are filled in with information from the DCB macro instruction, the information specified on the FILEDEF command, or, if the data set already exists, the data set label. However, if more than one source specifies information for a particular field, only one source is used.

The DCB fields are filled in in this order:

1. The DCB macro instruction in your program.

2. The fields you had specified on the FILEDEF command.

3. The data set label if the data set already exists.

The DCB macro instruction takes precedence over the FILEDEF and the data set label. Data set label information from an existing CMS file is used only when the OPEN is for input or update, otherwise the OPEN routine erases the existing file.

You can modify any DCB field either before the data set is opened or through a Data Control Block open exit. CMS supports only the Data Control Block exit of the EXIT LIST (EXLST) options.

When the data set is closed, the Data Control Block is restored to its original condition. Fields that were merged in at OPEN time from the FILEDEF and the data set label are cleared.

To accomplish this simulation, CMS supports certain essential macros for the following access methods:

- BDAM (direct) -- identifying a record by a key or by its relative position within the data set.

- BPAM (partitioned) -- seeking a named member within data set.

- BSAM/QSAM (sequential) -- accessing a record in a sequence in relation to preceding or following records.

- VSAM (direct or sequential) -- accessing a record sequentially or directly by key or address.

  **Note:** CMS support of OS VSAM files is based on VSE/VSAM. Therefore, the OS user is restricted to those functions available under VSE/VSAM. See the section "CMS Support for OS and VSE/VSAM Functions" for details.

CMS also updates those portions of the OS control blocks that are needed by the OS simulation routines to support a program during execution. Most of the simulated supervisory OS control blocks are contained in the following two CMS control blocks:

CMSCVT
: simulates the communication vector table. Location 16 contains the address of the CVT control section.

CMSCB
: is allocated from system free storage whenever a FILEDEF command or an OPEN (SVC 19) is issued for a data set. The CMS Control Block consists of a file control block (FCB) for the data file, and partial simulation of the job file control block (JFCB), input/output block (IOB), and data extent block (DEB).

The data control block (DCB) and the data event control block (DECB) are used by the access method simulation routines of CMS.

**Note:** The results may be unpredictable if two DCBs access the same data set at the same time. The GET and PUT macros are not supported for use with spanned records except in GET locate mode. READ, WRITE, and GET (in locate mode) are supported for spanned records, provided the filemode number is 4 and the data set is in physical sequential format.

GET (QSAM)
: All the QSAM options of GET are supported. Substitute mode is handled the same as move mode. If the DCBRECFM is FB, the filemode number is 4, and the last block is a short block, an EOF indicator (X'61FFFF61') must be present in the last block after the last record. Issue an explicit CLOSE prior to returning to CMS to obtain the last record when LOCATE mode is used with PUT.

GET (QISAM)
: QISAM is not supported in CMS.

PUT (QSAM)
: All the QSAM options of PUT are supported. Substitute mode is handled the same as move mode. If the DCBRECFM is FB, the filemode number is 4, and the last block is a short block, an EOF indicator is written in the last block after the last record. When LOCATE mode is used with PUT, issue an explicit CLOSE prior to returning to CMS to obtain the last record.

  The GET and PUT macros are not supported for use with spanned records except in GET locate mode. READ, WRITE, and GET (in locate mode) are supported for spanned records, provided the filemode number is 4, and the data set is physical sequential format.

PUT (QISAM)
: QISAM is not supported in CMS.

PUTX
: PUTX support is provided only for data sets opened for QSAM-UPDATE with simple buffering. READ/WRITE (BISAM) BISAM is not supported in CMS.

READ/WRITE (BSAM and BPAM)
: All the BSAM and BPAM options of READ and WRITE are supported except for the SE option (read backwards). READ (Offset Read of Keyed BDAM dataset) This type of READ is not supported because it is used only for spanned records.

READ/WRITE (BDAM)
: All the BDAM and BSAM (create) options of READ and WRITE are supported except for the R and RU options.

When an input or output error occurs, do not depend on OS sense bytes. An error code is supplied by CMS in the ECB in place of the sense bytes. These error codes differ for various types of devices and their meaning can be found in *VM/SP System Messages and Codes,* under DMS message 120S.

## BDAM Restrictions

The four methods of accessing BDAM records are:

1. Relative Block RRR
2. Relative Track TTR
3. Relative Track and Key TTK
4. Actual Address MBBCCHHR

The restrictions on these access methods are as follows:

- Only the BDAM identifiers underlined above can be used to refer to records, since CMS files have a two-byte record identifier.
- CMS BDAM files are always created with 255 records on the first logical track, and 256 records on all other logical tracks, regardless of the block size. If BDAM methods 2, 3, or 4 are used and the RECFM is U or V, the BDAM user must either write 255 records on the first track and 256 records on every track thereafter, or he must not update the track indicator until a NO SPACE FOUND message is returned on a write. For method 3 (WRITE ADD), this message occurs when no more dummy records can be found on a WRITE request. For methods 2 and 4, this does not occur, and the track indicator is updated only when the record indicator reaches 256 and overflows into the track indicator.
- The user must create variable length BDAM files (in PL/I they are Regional 3 files) entirely under CMS. He must also specify, on the XTENT option of the FILEDEF command, the exact number of records to be written. When reading variable length BDAM files, the XTENT and KEYLEN information specified for the file must duplicate the information specified when the file was created. CMS does not support WRITE ADD of variable length BDAM files; that is, the user cannot add additional records to the end of an already existing variable length BDAM file.
- Two files of the same filetype, both of which use keys, cannot be open at the same time. If a program that is updating keys does not close the file it is updating for some reason, such as a system failure or another IPL operation, the original keys for files that are not fixed format are saved in a temporary file with the same filetype and a filename of $KEYSAVE. To finish the update, run the program again.
- Variable length BDAM files must be created under CMS in their entirety, with the XTENT option of FILEDEF specifying the exact number of records to be written. When reading variable BDAM files, the XTENT and key length information specified must duplicate that created at file creation time. CMS does not support adding variable length records to BDAM files.

- Once a file is created using keys, additions to the file must not be made without using keys and specifying the original length.

- Note that there is limited support from the CMS file system for BDAM-created files (sparse). Sparse files are manipulated with CMS commands but are not treated as sparse files by most CMS commands. The number of records in the FST are treated as a valid record number.

- The number of records in the data set extent must be specified using the FILEDEF command. The default size is 50 records.

- The minimum LRECL for a CMS BDAM file with keys is eight bytes.

## Reading OS Data Sets and VSE Files Using OS Macros

CMS users can read OS sequential and partitioned data sets that that reside on OS disks. The CMS MOVEFILE command can be used to manipulate those data sets, and the OS QSAM, BPAM macros can be executed under CMS to read them.

The CMS MOVEFILE command can be used to manipulate and read VSE sequential files that reside on DOS disks. OS macros, however, can only be used to read sequential files from DOS formatted CKD disks. OS macros are *not* supported for reading sequential files on DOS formatted FB-512 disks.

The following OS Release 20.0 BSAM, BPAM, and QSAM macros can be used with CMS to read OS data sets and DOS files:

| | | |
|---|---|---|
| BLDL | ENQ | RDJFCB |
| BSP | FIND | READ |
| CHECK | GET | SYNADAF |
| CLOSE | NOTE | SYNADRLS |
| DEQ | POINT | WAIT |
| DEVTYPE | POST | |

CMS supports the following disk formats for the OS and OS/VS sequential and partitioned access methods:

- Split cylinders
- User labels
- Track overflow
- Alternate tracks

As in OS, the CMS support of the BSP macro produces a return code of 4 when attempting to backspace over a tape mark or when a beginning of an extent is found on an OS data set or a VSE file. If the data set or file contains split cylinders, an attempt to backspace within an extent, resulting in a cylinder switch, also produces a return code of 4.

### The ACCESS Command

Before CMS can read an OS data set or VSE file that resides on a non-CMS disk, you must issue the CMS ACCESS command to make the disk on which it resides available to CMS. The format of the ACCESS command can be found in the *VM/SP CMS Command and Macro Reference*. You must not specify options or file identification when accessing an OS or DOS disk.

You then issue the FILEDEF command to assign a CMS file identification to the OS data set or VSE file so that CMS can read it. The complete format of the FILEDEF command is found in the *VM/SP CMS Command and Macro Reference.* If you are issuing a FILEDEF for a VSE file, note that the OS program that will use the VSE file must have a DCB for it. For "ddname" in the FILEDEF command line, use the ddname in that DCB. With the DSN operand, enter the file-id of the VSE file.

Sometimes, CMS issues the FILEDEF command for you. Although the CMS MOVEFILE command, the supported CMS program interfaces, and the CMS OPEN routine each issue a default FILEDEF, you should issue the FILEDEF command yourself to ensure the appropriate file is defined.

After you have issued the ACCESS and FILEDEF commands for an OS sequential or partitioned data set or OS sequential file, CMS commands (such as ASSEMBLE and STATE) can refer to the OS data set or VSE file just as if it were a CMS file.

Several other CMS commands can be used with OS data sets and VSE files that do not reside on CMS disks. See the *VM/SP CMS Command and Macro Reference* for a complete description of the CMS ACCESS, FILEDEF, LISTDS, MOVEFILE, QUERY, RELEASE, and STATE commands.

For restrictions on reading OS data sets and VSE files under CMS, see the *VM/SP Planning Guide and Reference.*

The CMS FILEDEF command allows you to specify the I/O device and the file characteristics to be used by a program at execution time. In conjunction with the OS simulation scheme, FILEDEF simulates the functions of the data definition JCL statement.

FILEDEF may be used only with programs using OS macros and functions. For example:

```
filedef file1 disk proga data a1
```

After issuing this command, your program referring to FILE1 would access PROGA DATA on your A-disk.

If you wished to supply data from your terminal for FILE1, you could issue the command:

```
filedef file1 terminal
```

and enter the data for your program without recompiling.

```
fi tapein tap2 (recfm fb lrecl 50 block 100 9track den 800)
```

After issuing this command, programs referring to TAPEIN will access a tape at virtual address 182. (Each tape unit in the CMS environment has a symbolic name associated with it.) The tape must have been previously attached to the virtual machine by the VM/SP operator.

### The AUXPROC Option of the FILEDEF Command

The AUXPROC option can only be used by a program call to FILEDEF and not from the terminal. The CMS language interface programs use this feature for special I/O handling of certain (utility) data sets.

The AUXPROC option, followed by a fullword address of an auxiliary processing routine, allows that routine to receive control from DMSSEB before any device I/O is performed. At the completion of its processing, the auxiliary routine returns control to DMSSEB signaling whether or not I/O has been performed. If it has not been done, DMSSEB performs the appropriate device I/O.

When control is received from DMSSEB, the general-purpose registers contain the following information:

| | |
|---|---|
| GPR2 | = Data Control Block (DCB) address |
| GPR3 | = Base register for DMSSEB |
| GPR8 | = CMS OPSECT address |
| GPR11 | = File Control Block (FCB) address |
| GPR14 | = Return address in DMSSEB |
| GPR15 | = Auxiliary processing routine address |
| all other registers | = Work registers |

The auxiliary processing routine must provide a save area in which to save the general registers; this routine must also perform the save operation. DMSSEB does not provide the address of a save area in general register 13, as is usually the case. When control returns to DMSSEB, the general registers must be restored to their original values. Control is returned to DMSSEB by branching to the address contained in general register 14.

GPR15 is used by the auxiliary processing routine to inform to DMSSEB of the action that has been or should be taken with the data block as follows:

| **Register Content** | **Action** |
|---|---|
| GPR15=0 | No I/O performed by AUXPROC routine; DMSSEB will perform I/O. |
| GPR15<0 | I/O performed by AUXPROC routine and error was encountered. DMSSEB will take error action. |
| GPR15>0 | I/O performed by AUXPROC routine with residual count in GPR15; DMSSEB returns normally. |
| GPR15=64K | I/O performed by AUXPROC routine with zero residual count. |

# VSE Support Under CMS

CMS supports interactive program development for VSE. This includes creating, compiling, testing, debugging, and executing commercial application programs. The VSE programs can be executed in a CMS virtual machine or in a CMS Batch Facility virtual machine.

VSE files and libraries can be read under CMS. VSAM data sets can be read and written under CMS.

The CMS/DOS environment (called CMS/DOS) provides many of the same facilities that are available in VSE. However, CMS/DOS supports only those facilities that are supported by a single (background) partition. The VSE facilities supported by CMS/DOS are:

- VSE linkage editor
- Fetch support
- VSE Supervisor and I/O macros
- VSE Supervisor control block support
- Transient area support
- VSE/VSAM macros

This environment is entered each time the CMS SET DOS ON command is issued; VSAM functions are available in CMS/DOS only if the SET DOS ON (VSAM) command is issued. In the CMS/DOS environment, CMS supports many VSE facilities, but does not support OS simulation. When you no longer need VSE support under CMS, you issue the SET DOS OFF command and VSE facilities are no longer available.

CMS/DOS can execute programs that use the sequential access method (SAM) and virtual storage access method (VSAM), and can access VSE libraries.

CMS/DOS cannot execute programs that have execution-time restrictions, such as programs that use teleprocessing access methods or multitasking. DOS/VS COBOL, DOS PL/I, DOS/VS RPG II, and Assembler language programs are executable under CMS/DOS.

All of the CP and CMS online debugging and testing facilities (such as the CP ADSTOP and STORE commands and the CMS DEBUG environment) are supported in the CMS/DOS environment. Also, CP disk error recording and recovery is supported in CMS/DOS.

With its support of a CMS/DOS environment, CMS becomes an important tool for VSE application program development. Because CMS/DOS is designed as a VSE program development tool, it assumes in many cases that a VSE system exists, and uses it. The following sections describe what is supported, and what is not.

## Hardware Devices Supported

CMS/DOS routines can read real DOS disks containing VSE data files and VSE private and system libraries. This read support is limited to the following disks supported by VSE:

- IBM 2314 Direct Access Storage Facility
- IBM 2319 Disk Storage
- IBM 3310 Direct Access Storage
- IBM 3330 Disk Storage, Models 1 and 2
- IBM 3330 Disk Storage, Model 11
- IBM 3340 Direct Access Storage Facility
- IBM 3344 Direct Access Storage
- IBM 3350 Direct Access Storage
- IBM 3370 Direct Access Storage
- IBM 3375 Direct Access Storage

The following devices, which are supported by VSE, are not supported by CMS/DOS:

- Card Readers: 1442, 2560P, 2560S, 2596, 3504, 5425P, and 5425S
- Printers: 2560P, 2560S, 3203 Models 1 and 2, 3525, 5203, 5425P, and 5425S
- Disks: 2311

Also, CMS uses the CP spooling facilities and does not support dedicated unit record devices. Each CMS virtual machine supports only one virtual console, one reader, one punch, one printer, four tapes, and 26 disks. Programs that are executed in CMS/DOS are limited to the number of devices supported by CMS.

## CMS Support of VSE Functions

In addition to the CMS SET command used to invoke the CMS/DOS environment, there are a number of CMS/DOS commands and CMS commands with special CMS/DOS operands that provide CMS support of the following VSE functions:

- Assignment of logical units to particular physical devices.
- Associating VSE files with particular logical units.
- VSE librarian services.
- Compilation and testing of DOS/VS COBOL, DOS PL/I and RPGII programs.
- Execution of DOS/VS COBOL, DOS PL/I, and RPGII programs.

Figure 43 on page 388 summarizes these commands and operands. A detailed description and command format can be found in the *VM/SP CMS Command and Macro Reference*.

| Command | Operand | Comments |
|---|---|---|
| ASSGN | | Executable only in the CMS/DOS environment. Assigns CMS/DOS system or programmer logical units to a virtual device. |
| DLBL | | Defines a VSE or VSAM ddname and relates the ddname to a disk file. |
| DOSLIB | | Deletes, compacts, or lists information about the phases in a CMS/DOS phase library. |
| DOSLKED | | Executable only in the CMS/DOS environment. Link-edits CMS text file, or object modules from a VSE relocatable library, and places them in executable forms in a CMS/DOS phase library. |
| DOSPLI | | Executable only in the CMS/DOS environment. Compiles DOS PL/I source programs. |
| DSERV | | Executable only in the CMS/DOS environment. Displays information about VSE core image, relocatable, source statement, and procedure and/or transient directories. |
| ESERV | | Executable only in the CMS/DOS environment. Displays, updates, punches, or prints edited (E sublibrary) VSE source statement books. |
| FCOBOL | | Executable only in the CMS/DOS environment. Compiles DOS/VS COBOL source programs. |
| FETCH | | Executable only in the CMS/DOS environment. Fetches a CMS/DOS executable phase. |
| GENMOD | OS<br>DOS<br>ALL | Specifies the type of macro support needed to execute a module. The ALL operand is intended for CMS internal use. |
| GLOBAL | DOSLIB | The GLOBAL command can now specify CMS/DOS phase libraries, as well as text and macro libraries. |
| LISTIO | | Executable only in the CMS/DOS environment. Display information about CMS/DOS system and programmer logical units. |
| LOADMOD | | Checks that a module generated to execute in a specific macro simulation environment (CMS/DOS or CMS) is in the correct environment. |

**Figure 43 (Part 1 of 2). Summary of Changes to CMS Commands to Support CMS/DOS**

| Command | Operand | Comments |
|---|---|---|
| OPTION | | Executable only in the CMS/DOS environment. Sets compiler options for DOS/VS COBOL. |
| PSERV | | Executable only in the CMS/DOS environment. Copies and displays procedures in the VSE procedure libraries and/or spools the procedures to the CMS virtual printer and/or punch. |
| QUERY | UPSI | Executable only in the CMS/DOS environment. Displays current setting of CMS/DOS UPSI byte. |
| | OPTION | Executable only in the CMS/DOS environment. Displays CMS/DOS compiler options. |
| | DOSLNCNT | Displays the current number of SYSLST lines per page. |
| | DOS | Displays the current status (active or not active) of CMS/DOS. |
| | DOSLIB | Displays the names of all CMS/DOS phase libraries currently being searched for executable phases. |
| | LIBRARY | Displays the names of all CMS/DOS phase libraries to be searched, in addition to the text and macro libraries. |
| RSERV | | Executable only in the CMS/DOS environment. Copies and/or displays modules in a VSE relocatable library. Output can also be directed to the virtual printer or punch. |
| SET | DOS | Makes the CMS/DOS environment active or not active. |
| | DOSLNCNT nn | Specifies the number of SYSLST lines per page. |
| | UPSI | Executable only in the CMS/DOS environment. Sets the CMS/DOS UPSI byte. |
| SSERV | | Executable only in the CMS/DOS environment. Copies or displays books from the VSE source statement library. Output can also be directed to the virtual printer or punch. |

**Figure 43 (Part 2 of 2). Summary of Changes to CMS Commands to Support CMS/DOS**

## *Logical Unit Assignment*

A logical unit is a symbolic name by which a program may refer to a real I/O device without knowing the device address. Two examples of logical units are SYSRDR and SYSPCH.

The VSE supervisor uses two control blocks, the logical unit block (LUB) and the physical unit block (PUB), to map the symbolic name to the real device address. An entry in the LUB table for a particular logical unit, such as SYSRDR, contains a pointer to a PUB table entry. The PUB entry contains the address of the reader, X'00C'. Thus, all programs that read from the logical unit SYSRDR actually read from the device at address X'00C'.

On a real VSE machine, logical unit assignments are made dynamically via the ASSGN job statement or the ASSGN operator command. When using CMS/DOS, the CMS ASSGN command performs a similar function.

The ASSGN command in CMS/DOS assigns (or unassigns) a system or programmer logical unit to (or from) a virtual I/O device. If a disk is being assigned to a logical unit, the disk must have been previously accessed via the ACCESS command. As in VSE, you are not allowed to assign the system residence volume via the ASSGN command.

SYSLOG is the default value assigned to the terminal when SET DOS ON is issued.

The valid system logical units that can be assigned are:

| | | |
|---|---|---|
| SYSRDR | SYSLOG | SYSRLB |
| SYSIPT | SYSIN | SYSCAT |
| SYSPCH | SYSOUT | SYSCLB |
| SYSLST | SYSSLB | |

Other VSE system logical units cannot be assigned. The following VSE system logical units cannot be assigned to a VSE formatted FB-512 device:

SYSIN SYSIPT SYSRDR SYSLST SYSPCH

An error message is issued and the command terminated if any of the unsupported system logical units are specified in the ASSGN command. If SYSIN is specified, both the SYSIPT and SYSRDR LUB and PUB entries are filled in. If SYSOUT is specified, both the SYSLST and SYSPCH LUB and PUB entries are filled in.

If you wish to use VSE private relocatable, core image or source statement libraries, you must assign SYSRLB, SYSCLB or SYSSLB, respectively.

You can assign programmer units SYS000 through SYS241 with the ASSGN command. This deviates from VSE where the number of programmer logical units varies according to the number of partitions.

ASSGN creates a VSE Logical Unit Block (LUB) and Physical Unit Block (PUB) entry if the device is unassigned or alters the existing LUB/PUB relationship if the device is already assigned. ASSGN fills in a one-byte index in the LUB, which points to the proper PUB entry. This PUB entry contains the channel, unit, and device type information.

When a system or programmer logical unit is assigned to READER, PUNCH, or PRINTER, the reference is to a spooled unit record device. Card reader and terminal I/O data must not be blocked.

The ASSGN command is also used to ignore (IGN) or unassign (UA) a logical unit. An I/O operation for a logical unit that is in IGN status is effectively a NOP. When a logical unit is unassigned, its pointer to the PUB table is removed.

### Compiler Input/Output Assignments

The compilers supported by CMS/DOS expect input/output to be assigned to the following devices:

- SYSIN/SYSIPT must be assigned to the device where the input source file resides. Valid device types are reader, tape, or disk.
- The user should assign the following logical units to any of the indicated device types:

  SYSPCH to tape, punch, disk, or IGN

  SYSLST to tape, printer, disk, or IGN

  SYSLOG to terminal

  SYS001, SYS002, and SYS006 to disk.

  SYS003-SYS005 to tape or disk.

The maximum number of work files is six for DOS/VS COBOL Compiler (FCOBOL) and two for DOS PL/I Optimizing Compiler (DOSPLI).

You must assign SYSIN/SYSIPT. If it is unassigned at compilation time, an error message is issued and the FCOBOL or DOSPLI command is terminated.

If SYSPCH or SYSLST are unassigned at compilation time, the FCOBOL or DOSPLI EXEC file directs output to the disk where SYSIN resides if SYSIN is assigned to a read/write CMS disk. Otherwise, output is directed to the CMS read/write disk with the most read/write space. If SYSLOG is unassigned, it is assigned to the terminal. If SYS001 through SYSnnn are unassigned, output is directed to the CMS disk with the most read/write space.

#### Interrogating I/O Assignments

The current I/O assignments may be displayed on the terminal by entering the CMS/DOS LISTIO command. You can selectively display the system and/or programmer logical units as a group or as a specific unit. With the EXEC option of the LISTIO command you can create a disk file containing the list of assignments.

## VSE Supervisor and I/O Macros Supported by CMS/DOS

CMS/DOS supports the VSE Supervisor macros and the SAM and VSAM I/O macros to the extent necessary to execute the DOS/VS COBOL Compiler, the DOS PL/I Optimizing Compiler, and DOS/VS RPG II Compiler under CMS/DOS. CMS/DOS supports VSE Supervisor macros described in the publication *VSE Macro Reference.*

Since CMS is a single-user system executing in a virtual machine with virtual storage, VSE operations, such as multitasking, that cannot be simulated in CMS are ignored.

The following information deals with the type of support that CMS/DOS provides in the simulation of VSE Supervisor and Sequential Access Method I/O macros. For a discussion of VSAM macros, see the section "CMS Support for OS and VSE/VSAM Functions."

### *Supervisor Macros*

CMS/DOS supports physical IOCS macros and control program function macros for VSE. Figure 44 on page 392 lists the physical IOCS macros and describes their

support. Figure 45 on page 392 lists the control program function macros and their support. Refer to *VM/SP System Logic and Problem Determination Guide, Volume 2* for details of the macros' operation.

| Macro | Support |
|---|---|
| CCB (command control block) | The CCB is generated. |
| IORB (Input/Output Request Block) | Supported for DASD I/O. |
| EXCB (execute channel program) | The REAL operand is not supported; all other operands are supported. |
| WAIT | Supported. Issued whenever your program requires an I/O operation (started by an EXCP macro) to be completed before execution of program continues. |
| SECTVAL (sector value) | Supported for VSAM. See Figure 50 |
| OPEN/OPENR | Supported. Activates a data file. |
| LBRET (label processing return) | Not supported. |
| FEOV (forced end of volume) | Not supported. |
| SEOV (system end of volume) | Not supported. |
| CLOSE/CLOSER | Supported. Deactivates a data file. |

**Figure 44. Physical IOCS Macros Supported by CMS/DOS**

| Function/ Macro | SVC. No. Dec | Hex | Support |
|---|---|---|---|
| EXCP | 0 | 0 | Used to read from CMS or DOS/OS formatted Disks. |
| FETCH | 1 | 1 | Used to bring a problem program phase into user storage and to start execution of the phase if the phase was found. Operand SYS=YES is not supported. |
| FETCH | 2 | 2 | Used to bring a $$B-transient phase into the CMS transient area (or if the phase is in the CMSDOS segment, not to load it), and start execution of the phase if the phase was found. Operand SYS=YES is not supported. |
| FORCE DEQUEUE | 3 | 3 | Not supported, see note 2. |

**Figure 45 (Part 1 of 9). SVC Support Routines and Their Operation**

| Function/ Macro | SVC. No. Dec Hex | Support |
|---|---|---|
| LOAD | 4 4 | Used to bring a problem program phase into user storage, and return the caller the entry point address of the phase just loaded. Operand SYS=YES is not supported. |
| MVCOM | 5 5 | Provides the user with a means of altering positions 12 through 23 of the partition communications region (BGCOM). |
| CANCEL | 6 6 | Cancels a VSE session either by a VSE program request, or by request from any of the CMS routines handling CMS/DOS. |
| WAIT | 7 7 | Used to wait on a CCB, IORB, ECB, or TECB. (Note that CMS/DOS does not support ECB's or TECB's). In the case of CCB's, they are always posted by the DMSXCP routine before returning to the caller.<br><br>The WAIT support under CMS/DOS will effectively be a branch to the CMS/DOS POST routine. |
| CONTROL | 8 8 | Temporarily return control from a $$B-transient to the problem program. |
| LBRET | 9 9 | Return to the $$B-transient after an SVC 8 was issued to give control to the problem program. |
| SET TIMER | 10 A | No operation, successful return code of 0 is given in R15. See note 1. |
| TRANS. RETURN | 11 B | Return from a $$B-transient to the calling problem program. |
| JOB CONTROL 'AND' | 12 C | Resets flags to 0 in the linkage control byte in BGCOM (communication region). If R1 = 0, SVC 12 has another meaning. Bit 5 of JCSW4 (CONREG byte 59) is turned off., |
| JC FLAGS | 13 D | Not supported, see note 2. |
| EOJ | 14 E | Normally terminates execution of a problem program. |
| SYSIO | 15 F | Not supported, see note 2. |
| PC STXIT | 16 10 | Establish or terminate linkage to a user's program check routine. |
| PC EXIT | 17 11 | Used to provide supervisory support for the EXIT macro. SVC 17 provides a return from the user's PC routine to the next sequential instruction in the program that was interrupted due to a program check. |

**Figure 45 (Part 2 of 9). SVC Support Routines and Their Operation**

| Function/ Macro | SVC. No. Dec Hex | Support |
|---|---|---|
| IT STXIT | 18 12 | No operation, successful return code of 0 is given in R15. See note 1. |
| IT EXIT | 19 13 | Not supported, see note 2. |
| OC STIXIT | 20 14 | No operation, successful return code of 0 is given in R15. See note 1. |
| OC EXIT | 21 15 | Not supported, see note 2. |
| SIEZE | 22 16 | No operation, successful return code of 0 is given in R15. See note 1. |
| LOAD HEADER | 23 17 | Not supported, see note 2. |
| SETIME | 24 18 | No operation, successful return code of 0 is given in R15. See note 1. |
| HALT I/O | 25 19<br><br>26 1A | Not supported, see note 2.<br><br>Validate address limits. The upper address must be specified in general register 2 and the lower address must be specified in general register 1. |
| TP HALT I/O | 27 1B | Not supported, see note 2. |
| MR EXIT | 28 1C | Not supported, see note 2. |
| WAITM | 29 1D | Not supported, see note 2. |
| QWAIT | 30 1E | Not supported, see note 2. |
| QPOST | 31 1F | Not supported, see note 2. |
| | 32 20 | Reserved |
| COMRG | 33 21 | Used to provide the caller with the address of the partition communications region.<br><br>DMSDOS provides the caller with the address of the partition communications region, in the user's register 1. |
| GETIME | 34 22 | Provides support for the GETIME macro. SVC 34 updates the date field in the communications region. The GMT operand is not supported. |
| HOLD | 35 23 | No operation, successful return code of 0 is given in R15. See note 1. |
| FREE | 36 24 | No operation, successful return code of 0 is given in R15. See note 1. |
| AB STXIT | 37 25 | Establish or terminate linkage to a user's abnormal termination routine. Supported for OPINION=DUMP or NODUMP. |
| ATTACH | 38 26 | Not supported, see note 2. |
| DETACH | 39 27 | Not supported, see note 2. |

**Figure 45 (Part 3 of 9). SVC Support Routines and Their Operation**

| Function/ Macro | SVC. No. Dec Hex | Support |
|---|---|---|
| POST | 40 28 | Used to post an ECB, IORB, TECB, or CCB. Byte 2, bit 0 of the specified control block are turned 'on' by DMSDOS. |
| DEQ | 41 29 | No operation, successful return code of 0 is given in R15. See note 1. |
| ENQ | 42 2A | No operation, successful return code of 0 is given in R15. See note 1. |
| | 43 2B | Reserved |
| UNIT CHECKS | 44 2C | Not supported, see note 2. |
| EMULATOR INTERF. | 45 2D | Not supported, see note 2. |
| OLTEP | 46 2E | Not supported, see note 2. |
| WAITF | 47 2F | Not supported, see note 2. |
| CRT TRANS | 48 30 | Not supported, see note 2. |
| CHANNEL PROG. | 49 31 | Not supported, see note 2. |
| LIOCS DIAG. | 50 32 | Issued by a logical IOCS routine when the LIOCS is called to perform an operation for which the LIOCS was not generated to perform.<br><br>The error message "unsupported function in a LIOCS routine" is issued, and the session is then terminated. |
| RETURN HEADER | 51 33 | Not supported, see note 2. |
| TTIMER | 52 34 | No operation, successful return code of 0 is given in R15. See note 1. RO is also cleared. |
| VTAM EXIT | 53 35 | Not supported, see note 2. |
| FREEREAL | 54 36 | Not supported, see note 2. |
| GETREAL | 55 37 | Not supported, see note 2. |
| POWER | 56 38 | Not supported, see note 2. |
| POWER | 57 39 | Not supported, see note 2. |
| SUPVR. INTERF. | 58 3A | Not supported, see note 2. |
| EOJ INTERF. | 59 3B | Not supported, see note 2. |
| GETADR | 60 3C | Not supported, see note 2. |
| GETVIS | 61 3D | Used to obtain free storage for scratch use or for obtaining an area into which a relocatable program may be loaded. The PAGE, POOL, and SVA GETVIS options are ignored. |
| FREEVIS | 62 3E | Used to return the free storage obtained via an earlier GETVIS call. |

**Figure 45 (Part 4 of 9). SVC Support Routines and Their Operation**

| Function/ Macro | SVC. No. Dec Hex | Support |
|---|---|---|
| USE | 63 3F | The USE/RELEASE function has been replaced by SVC 110 (LOCK/UNLOCK) for serially controlling system resources. All SVC 63 and 64 requests are mapped into SVC 110 requests respectively. Return codes previously associated with USE/RELEASE under CMS/DOS are maintained. |
| RELEASE | 64 40 | Reference SVC 63. |
| CDLOAD | 65 41 | Used to load a relocatable phase into storage, unless the program has already been loaded. |
| RUNMODE | 66 42 | Used by a problem program to find out if the program is running in real or virtual mode. The caller's register 0 is zeroed to indicate that the program is running in virtual mode. |
| PFIX | 67 43 | No operation, successful return code of 0 is given in R15. See note 1. |
| PFREE | 68 44 | No operation, successful return code of 0 is given in R15. See note 1. |
| REALAD | 69 45 | Not supported, see note 2. |
| VIRTAD | 70 46 | Not supported, see note 2. |
| SETPFA | 71 47 | No operation, successful return code of 0 is given in R15. See note 1. |
| GETCBUF/FREECBUF | 72 48 | Not supported, see note 2. |
| SETAPP | 73 49 | Not supported, see note 2. |
| PAGE FIX | 74 4A | Not supported, see note 2. |
| SECTVAL | 75 4B | Used by I/O routines to obtain a sector number for a 3330, 3330-11, 3340, or 3350 device. |
| SYSREC | 76 4C | Not supported, see note 2. |
| TRANSCCW | 77 4D | Not supported, see note 2. |
| CHAP | 78 4E | Not supported, see note 2. |
| SYNCH | 79 4F | Not supported, see note 2. |
| SETT | 80 50 | Not supported, see note 2. |
| TESTT | 81 51 | Not supported, see note 2. |
| LINKAGE | 82 52 | Not supported, see note 2. |
| ALLOCATE | 83 53 | Not supported, see note 2. |
| SET LIMIT | 84 54 | Not supported, see note 2. |

**Figure 45 (Part 5 of 9). SVC Support Routines and Their Operation**

| Function/ Macro | SVC. No. Dec Hex | Support |
|---|---|---|
| RELPAGE | 85 55 | Provides support for the RELPAG macro. At entry register 1 points to a list of 8-byte area. Each entry contains the beginning address and the length-1 of an area to be released. A non-zero byte following an entry indicates the end of the list. An area is released only if it contains at least a full CP page (4k bytes). Pages are released when the virtual machine calls CP via DIAGNOSE code X'10'. On return R15 holds return code as follows:<br>R15 = 0 all areas have been released<br>R15 = 2 one or more negative area lengths were specified<br>R15 = 4 one or more pages to be released were outside the user storage area<br>R15 =16 at least one entry contains a beginning address outside the user storage area. |
| FCEPGOUT | 86 56 | No operation, successful return code of 0 is given in R15. See note 1. |
| PAGEIN | 87 57 | No operation, successful return code of 0 is given in R15. See note 1. |
| TPIN | 88 58 | Not supported, see note 2. |
| TPOUT | 89 59 | Not supported, see note 2. |
| PUTACCT | 90 5A | Not supported, see note 2. |
| POWER | 91 5B | Not supported, see note 2. |
| XECBTAB | 92 5C | Not supported, see note 2. |
| XPOST | 93 5D | Not supported, see note 2. |
| XWAIT | 94 5E | Not supported, see note 2. |
| AB EXIT | 95 5F | Exit from abnormal task termination routine and continue the task. |
| TT EXIT | 96 60 | Not supported, see note 2. |
| TT STXIT | 97 61 | Not supported, see note 2. |
| EXTRACT | 98 62 | Support for EXTRACT macro of VSE. The caller requests PUB information, CPUID, or storage boundary information. Register 1 on entry points to a parameter list. Output is placed in an area provided by caller. |

**Figure 45 (Part 6 of 9). SVC Support Routines and Their Operation**

| Function/ Macro | SVC. No. Dec Hex | Support |
|---|---|---|
| GETVCE | 99 63 | Support for GETVCE macro. Caller requests device information about specific DASD. Information is returned in an output area pointed to from the parameter list. Register 1 contains a pointer to the parameter list on entry. |
| | 100 64 | Reserved |
| MODVCE | 101 65 | No operation, successful return code of 0 is given in R15. See note 1. |
| | 102 66 | Reserved. |
| SYSFIL | 103 67 | Not supported, see note 2. |
| EXTENT | 104 68 | No operation, successful return code of 0 is given in R15. See note 1. |
| SUBSID | 105 69 | SUBSID.. the 'INQUIRY' function is supported for the supervisor sub-system. Information returned is described by the SUPSSID control block. The SUBSID 'NOTIFY' and 'REMOVE' functions are not supported. |
| LINKAGE | 106 6A | Not supported, see note 2. |

**Figure 45 (Part 7 of 9). SVC Support Routines and Their Operation**

| Function/ Macro | SVC. No. Dec Hex | Support |
|---|---|---|
| TASK INTERF. | 107 6B | Provides macro interface support for system information retrieval. The parameters supported are:<br>GETFLD:<br><br>Field=ppsavar — returns problem program save area address.<br><br>=savar — returns current save area address.<br><br>=maintask — returns maintask TID in R1.<br><br>=aclose — return in R1 1 if in process, 0 if not.<br><br>=pcexit — returns the pcexit routine address and save area in R0 and R1 respectively. If the exit routine is currently active, bit 0 in R0 is set ON. If no exit is defined, it returns a 0 in both R0 and R1.<br><br>MODFLD:<br><br>=vsamopen — set bit X'08' in tab tabflags byte if R1¬=0<br><br>=aclose — set bit X'10' in tab tabflags byte if R1¬=0<br><br>The MODFLD requests for fields CNCLALL and OPENSVA are treated as a NOP with a return code of 0.<br><br>All other GETFLD/MODFLD requests as well as all other SVC 107 macro calls are unsupported. The error message DMSGMF121S is issued and the request cancelled. See note 2. |
| DATA SECURE | 108 6C | Not supported, see note 2. |
| PAGESTAT | 109 6D | Not supported, see note 2. |

**Figure 45 (Part 8 of 9). SVC Support Routines and Their Operation**

| Function/ Macro | SVC. No. Dec Hex | Support |
|---|---|---|
| LOCK/UNLOCK | 110 6E | Used to control access to resources. Access is maintained in either a 'shared' or 'exclusive' control environment. Counters are maintained as well as the type of control for each resource in a table (LOCKTAB) built in free storage when DOS is SET ON. All entries not unlocked by the program are cleared at both normal and abnormal end-of-job.<br>All requests for resource control are passed to SVC 110 through the DTL macro (Define the Lock). SVC 63 requests are mapped into a dummy DTL and processed by SVC 110. |

**Figure 45 (Part 9 of 9). SVC Support Routines and Their Operation**

**Notes:**

1. No operation:
   In each case, register 15 is cleared to simulate successful operation, and all other registers are returned unchanged, unless otherwise noted.

2. Not supported:
   For unsupported SVCs, an error message is given, and the SVC is treated as a "cancel".

## *Sequential Access Method -- Declarative Macros*

CMS/DOS supports the following declarative macros:

- DTFCD - Types X'02' and X'04'
- DTFCN - Types X'03'
- DTFDI - Types X'33'
- DTFMT - Types X'10', X'11', X'12', and X'14'
- DTFPR - Types X'08'
- DTFSD - Types X'20'

The CDMOD, DIMOD, MTMOD, and PRMOD, macros generate the logical IOCS routines that correspond with the declarative macros. For files on disk, the logical IOCS routines used during program execution reside in the CMSBAM DCSS and are not generated within the program. The operands that CMS/DOS supports for the DTF are also supported for the xxMOD macro. In addition, CMS/DOS supports three internal macros that the COBOL and PL/I compilers require: DTFCP (types X'31' and X'32'), CPMOD, and DTFSL.

### DTFCD Macro -- Defines the File for a Card Reader

CMS/DOS does not support the ASOCFLE, FUNC, TYPEFILE=CMBND, and OUBLKSZ operands of the DTFCD macro. CMS/DOS ignores the SSELECT operand and any mode other than MODE=E. Figure 46 describes the DTFCD macro operands and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

| Operand | Status | Description |
|---|---|---|
| DEVADDR=SYSxxx | | Symbolic unit for reader-punch used for this file. |
| IOAREA1=xxxxxxxx | * | Name of the first I/O area. |
| ASOCFLE=xxxxxxxx | * | Not supported. |
| BLKSIZE=nnn | * | Length of one I/O area, in bytes. If omitted, 80 is assumed. If CTLCHR=YES is specified, BLKSIZE defaults to 81. |
| CONTROL=YES | | CNTRL macro used for this file. Omit CTLCHR for this file. Does not apply to 2501. |
| CRDERR=RETRY | * | Retry if punching error is detected. Applies to 2520 and 2540 only. However, this situation is never encountered under CMS/DOS because hardware errors are not passed to the LIOCS module. |
| CTLCHR=xxx | | (YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit CONTROL for this file. |
| DEVICE=nnnn | * | (2501, 2520, 2540, 3505, or 3525). If omitted, 2540 is default. |
| EOFADDR=xxxxxxxx | | Name of your end-of-file routine. |
| ERROPT=xxxxxx | * | IGNORE, SKIP, or name. Applies to 3505 and 3525 only. |
| FUNC=xxx | * | Not supported. |
| IOAREA2=xxxxxxxx | * | If two output areas are used, name of second area. |
| IOREG=(nn) | | Register number if two I/O areas were used and GET or PUT does not specify a work area. Omit WORKA. |
| MODE=xx | * | Only MODE=E is supported. |
| MODNAME=xxxxxxxx | | Name of the logic module that is used with the DTF table to process the file. |
| OUBLKSZ=nn | * | Not supported. |
| RDONLY=YES | * | Causes a read-only module to be generated. |
| RECFORM=xxxxxx | | (FIXUNB, VARUNB, UNDEF). If omitted, FIXUNB is default. |
| RECSIZE=(nn) | * | Register number if RECFORM=UNDEF. |
| SEPASMB=YES | | DTFCD is to be assembled separately. |
| SSELECT=n | * | Ignored. |
| TYPEFLE= | * | Input or output. |
| WORKA=YES | | I/O records are processed in work areas instead of the I/O areas. |

**Figure 46. CMS/DOS Support of DTFCD Macro**

### DTFCN Macro - Define the File for a Console

CMS/DOS supports all of the operands of the DTFCN macro. Figure 47 describes the operands of the DTFCN macro and their support under CMS/DOS. The status column is blank because the CMS/DOS and VSE support of DTFCN are the same.

| Operand | Status | Description |
|---|---|---|
| DEVADDR=SYSxxx | | Symbolic unit for the console used for this file. |
| IOAREA1=xxxxxxxx | | Name of I/O area. |
| BLKSIZE=nnn | | Length in bytes of I/O area (for PUTR macro usage, length of output part of I/O area). If RECFORM=UNDEF, maximum is 256. If omitted, 80 is default. |
| INPSIZE=nnn | | Length in bytes for input part of I/O area for PUTR macro usage. |
| MODNAME=xxxxxxxx | | Logic module name for this DTF. If omitted, IOCS generates a standard name.<br><br>The logic module is generated as part of the DTF. |
| RECFORM=xxxxxx | | (FIXUNB or UNDEF). If omitted, FIXUNB is default. |
| RECSIZE=(nn) | | Register number if RECFORM=UNDEF. General registers 2 through 12, enclosed in parentheses. |
| TYPEFLE=xxxxxx | | (INPUT, OUTPUT, or CMBND). Input processes both input and output. CMBND must be specified for PUTR macro usage. If omitted, INPUT is default. |
| WORKA=YES | | GET or PUT specifies work area. |

**Figure 47. CMS/DOS Support of DTFCN macro**

### DTFDI MACRO - Define the File for Device Independence for System Logical Units

CMS/DOS supports most operands of the DTFDI macro. Figure 48 describes the operands of the DTFDI macro and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

| | | |
|---|---|---|
| DEVADDR=SYSxxx | | (SYSIPT, SYSLST, SYSPCH, or SYSRDR). System logical unit. CMS/DOS issues an error message if the logical unit specified on the DTF does not match the logical unit specified on the corresponding DLBL command. |
| IOAREA1=xxxxxxxx | | Name of the first I/O area. |

**Figure 48 (Part 1 of 2). CMS/DOS Support of DTFDI Macro**

| | | |
|---|---|---|
| CISIZE=n | * | This operand specifies the control interval size for a DOS formatted FB-512 device assigned to a nonsystem file logical unit. This operand is ignored for count-key-data devices and CMS formatted disks. |
| EOFADDR=xxxxxxxx | | Name of your end-of-file routine. |
| FBA=YES | | This operand is not required and is ignored if specified. |
| ERROPT=xxxxxxxx | | (IGNORE, SKIP, or name of your error routine). Prevents termination on errors. |
| IOAREA2=xxxxxxxx | | If two I/O areas are used, name of second area. |
| IOREG2=(nn) | | Register number. If omitted and two I/O areas are used, register 2 is default. General registers 2 through 12, enclosed in parentheses. |
| MODNAME=xxxxxxxx | | DIMOD name for this DTF. If omitted, IOCS generates a standard name. This operand is ignored with DASD. The SAM OPEN routines within the CMSBAM DCSS always load an IBM supplied logic module and link it to the DTF. |
| RDONLY=YES | | Generates a read-only module. Requires a module save area for each routine using the module. |
| RECSIZE=nnn | | Number of characters in record. Default values: 121 (SYSLST), 81 (SYSPCH), 80 (other). |
| SEPASMB=YES | | DTFDI to be assembled separately. |
| TRC=YES | * | Not supported. |
| WLRERR=xxxxxxxx | | Name of your wrong-length record routine. |

**Figure 48 (Part 2 of 2). CMS/DOS Support of DTFDI Macro**

### DTFMT Macro -- Define the File for a Magnetic Tape

CMS/DOS does not support the ASCII, BUFOFF, HDRINFO, LENCHK, and READ=BACK operands of the DTFMT macro. Tape I/O operations are limited to reading in the forward direction.

You may use the FILABL operand in the DTFMT macro to specify that you have a standard tape label file, a nonstandard tape label file, or an unlabeled tape. The type of tape label processing depends on the option selected. See "Tape Labels in CMS" in the *VM/SP CMS User's Guide* for a complete description of tape label processing in CMS/DOS.

Figure 49 describes the DTFMT macro operands and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

| Operand | Status | Description |
|---|---|---|
| BLKSIZE=nnnnn | | Length of one I/O area in bytes (maximum = 32,767. |
| DEVADDR=SYSxxx | | Symbolic unit for tape drive used for this file. |
| EOFADDR=xxxxxxxx | | Name of your end-of-file routine. |
| FILABL=xxxx | | (NO, STD, or NSTD). If NSTD specified, include LABADDR. |
| IOAREA1=xxxxxxxx | | Name of first I/O area. |
| ASCII=YES | * | Not supported. |
| BUFOFF=nn | * | Not supported. |
| CKPTREC=YES | | Checkpoint records are interspersed with input data records. IOCS bypasses checkpoint records. |
| ERREXT=YES | | Additional errors and ERET are desired. |
| ERROPT=xxxxxxxx | | (IGNORE, SKIP, or name of error routine). Prevents job termination on error records. |
| HDRINFO=YES | * | Not supported. |
| IOAREA2=xxxxxxxx | | If two I/O areas are used, the name of the second area. |
| IOREG=(nn) | | Register number. Use only if GET or PUT does not specify a work area or if two I/O areas are used. Omit WORKA. General registers 2 through 12, enclosed in parentheses. |
| LABADDR=xxxxxxxx | | Name of your label routine if FILABL=NSTD, or if FILABL=STD and user-standard labels are processed. |
| LENCHK=YES | * | Not supported. |
| MODNAME=xxxxxxxx | | Name of MTMOD logic module for this DTF. If omitted, IOCS generates standard name. |
| NOTEPNT=xxxxxx | | (YES or POINTS). YES if NOTE, POINTW, POINTR, or POINTS macro used. POINTS if only POINTS macro used. |
| RDONLY=YES | | Generate read-only module. Requires a module save area for each routine using the module. |
| READ=xxxxxxx | * | CMS/DOS only supports READ=FORWARD. |
| RECFORM=xxxxxx | | (FIXUNB, FIXBLK, VARUNB, VARBLK, SPNUNB, SPNBLK, or UNDEF). For work files use FIXUNB or UNDEF. If omitted, FIXUNB is assumed. |

**Figure 49 (Part 1 of 2). CMS/DOS Support of DTFMT Macro**

| Operand | Status | Description |
|---|---|---|
| RECSIZE=nnnn | | If RECFORM=FIXBLK, number of characters in the record. If RECFORM=UNDEF, register number. Not required for other records. General registers 2 through 12, enclosed in parentheses. |
| REWIND=xxxxxx | | (UNLOAD or NORWD). Unload on CLOSE or end-of-volume, or prevent rewinding. If omitted, rewind only. |
| SEPASMB=YES | | DTFMT is to be assembled separately. |
| TPMARK=NO | | Prevent writing a tapemark ahead of data records if FILABL=NSTD or NO. |
| TYPEFLE=xxxxxx | | (INPUT, OUTPUT, or WORK). If omitted, INPUT is default. |
| VARBLD=(nn) | | Register number, if RECFORM=VARBLK and records are built in the output area. General registers 2 through 12 are enclosed in parentheses. |
| WLRERR=xxxxxxxx | | Name of wrong-length record routine. |
| WORKA=YES | | GET or PUT specifies a work area. Omit IOREG. |

**Figure 49 (Part 2 of 2). CMS/DOS Support of DTFMT Macro**

### DTFPR Macro - Define the File for a Printer

CMS/DOS does not support the ASOCFLE, ERROPT=IGNORE, and FUNC operands of the DTFPR macro. Figure 50 describes the operands of the DTFPR macro and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

| Operand | Status | Description |
|---|---|---|
| DEVADDR=SYSxxx | | Symbolic unit for the printer used for this file. |
| IOAREA1=xxxxxxxx | | Name for the first output area. |
| ASOCFLE=xxxxxxxx | * | Not supported. |
| BLKSIZE=nnn | * | Length of one output area, in bytes. If omitted, 121 is default. |
| CONTROL=YES | | CNTRL macro used for this file. Omit CTLCHR for this file. |
| CTLCHR=xxx | | (YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit CONTROL for this file. |

**Figure 50 (Part 1 of 2). CMS/DOS Support of DTFPR Macro**

| Operand | Status | Description |
|---|---|---|
| DEVICE=nnnn | * | (1403, 1443, 3203, or 3211). If omitted, 1403 is default. |
| ERROPT=xxxxxxxx | * | RETRY or the name of your error routine for 3211. Not allowed for other devices. IGNORE is not supported. |
| FUNC=xxxx | * | Not supported. |
| IOAREA2=xxxxxxxx | | If two output areas are used, name of second area. |
| IOREG=(nn) | | Register number; if two output areas used and GET or PUT does not specify a work area. Omit WORKA. |
| MODNAME=xxxxxxxx | | Name of PRMOD logic module for this DTF. If omitted, IOCS generates standard name. |
| PRINTOV=YES | | PRTOV macro used for this file. |
| RDONLY=YES | | Generate a read-only module. Requires a module save area for each routine using the module. |
| RECFORM=xxxxxx | | (FIXUNB, VARUNB, or UNDEF). If omitted, FIXUNB is default. |
| RECSIZE=(nn) | | Register number if RECFORM=UNDEF. |
| SEPASMB=YES | | DTFPR is to be assembled separately. |
| STLIST=YES | | Use 1403 selective tape listing feature. |
| TRC=YES | * | Not supported. |
| UCS=xxx | | (ON) process data checks. (OFF) ignores data checks. Only for printers with the UCS feature or 3203 or 3211. If omitted, OFF is default. |
| WORKA=YES | | PUT specifies work area. Omit IOREG. |

**Figure 50 (Part 2 of 2). CMS/DOS Support of DTFPR Macro**

### DTFSD Macro - Define the File for a Sequential DASD

CMS/DOS does not support the FEOVD, HOLD, and LABADDR operands of the DTFSD macro. Figure 51 describes the operands of the DTFSD macro and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

| Operand | Status | Description |
|---|---|---|
| BLKSIZE=nnnn | | Length of one I/O area, in bytes. |

**Figure 51 (Part 1 of 4). CMS/DOS Support of DTFSD Macro**

| Operand | Status | Description |
|---|---|---|
| CISIZE=n | * | This operand specifies the control interval size for a DOS formatted FB-512 device assigned to a nonsystem file logical unit. This operand is ignored for count-key-data devices and CMS formatted disks. |
| EOFADDR=xxxxxxxx | | Name of your end-of-file routine. |
| IOAREA1=xxxxxxxx | | Name of first I/O area. |
| CONTROL=YES | | This operand is ignored. CONTROL=YES is always included. |
| DELETFL=NO | * | If DELETFL=NO is specified, the work file is not erased. Otherwise, when the work file is closed, CMS/DOS erases it. |
| DEVADDR=SYSnnn | * | Symbolic unit. This operand is optional. If DEVADDR is not specified, all I/O requests are directed to the logical unit identified on the corresponding CMS/DOS DLBL command.<br><br>If a valid logical unit is specified with the DEVADDR operand of the DTF and a different, but also valid, logical unit is specified on the DLBL command, the unit specified on the DLBL command overrides the unit specified in the DTF. However, CMS/DOS issues an error message if a valid logical unit is specified in the DTF and no logical unit is specified on the corresponding DLBL command. |
| DEVICE=nnnn | * | This operand is ignored. The actual device type is determined by OPEN. |
| ERREXT=YES | | Additional error facilities and ERET are desired. This operand is ignored. ERREXT=YES is always included. |
| ERROPT=xxxxxxxx | | (IGNORE, SKIP, or name of error routine.) Prevents job termination on error records. Do not use SKIP for output files. |
| FEOVD=YES | * | Not supported. |
| HOLD=YES | * | Not supported. HOLD=YES is specified for DTFSD update or work files to provide a track hold capability. However, the CMS/DOS open routine sets the track hold bit off and bypasses track hold processing. |

**Figure 51 (Part 2 of 4). CMS/DOS Support of DTFSD Macro**

| Operand | Status | Description |
|---|---|---|
| IOAREA2=xxxxxxxx | | If two I/O areas are used, name of second area. |
| IOREG=(nn) | | Register number. Use only if GET or PUT does not specify work area or if two I/O areas are used. Omit WORKA. |
| LABADDR=xxxxxxxx | * | Not supported. |
| MODNAME=xxxxxxxx | | This operand is not required. If specified, it is ignored. The SAM OPEN routines within the CMSBAM DCSS always load an IBM supplied logic module and link it to the DTF. |
| NOTEPNT=xxxxxxxx | | Indicates that NOTE, POINTR, POINTW, and POINTS are used. This operand is ignored. NOTEPNT=YES is always included. |
| RDONLY=YES | | This operand is not required and is ignored if specified. RDONLY=YES is always included. |
| PWRITE=YES | * | For a DOS formatted FB-512 disk, this operand specifies that for output operations a physical write occurs for every logical block. This operand is ignored for count-key-data devices and CMS formatted disks. DOS formatted FB-512 disks are not supported for output. |
| RECFORM=xxxxxx | | (FIXUNB, FIXBLK, VARUNB, SPNUNB, SPNBLK, VARBLK, or UNDEF). If omitted, FIXUNB is assumed.<br><br>For work files use FIXUNB or UNDEF. Although work files contain fixed-length unblocked records, the CMS file system handles work UNDEF files as variable-length record files. If you specify FIXBLK, VARBLK, or UNDEF when creating a CMS file on a CMS CMS disk, CMS writes the file in variable-length format. The LISTFILE command would show the file as V format. If you specify FIXUNB when creating a CMS file on a CMS disk, CMS writes the file in fixed-length format. |
| RECSIZE=nnnnn | | If RECFORM=FIXBLK, number of characters in record. If RECFORM=SPNUNB, SPNBLK, or UNDEF, register number. Not required for other records. |
| SEPASMB=YES | | DTFSD is to be assembled separately. |

**Figure 51 (Part 3 of 4). CMS/DOS Support of DTFSD Macro**

| Operand | Status | Description |
|---|---|---|
| TRUNCS=YES | | RECFORM=FIXBLK or TRUNC macro used for this file. |
| TYPEFLE=xxxxxx | | (INPUT, OUTPUT, or WORK). If omitted, INPUT is assumed. |
| UPDATE=YES | | Input file or work file is to be updated. |
| VARBLD=(nn) | | Register number if RECFORM=VARBLK and records are built in the output area. Omit if WORKA=YES. |
| VERIFY=YES | | Check disk records after they are written. |
| WLRERR=xxxxxxxx | | Name of your wrong-length record routine. |
| WORKA=YES | | GET or PUT specifies work area. Omit IOREG. Required for RECFORM=SPNUNB or SPNBLK. |

**Figure 51 (Part 4 of 4). CMS/DOS Support of DTFSD Macro**

### *Sequential Access Method -- Imperative Macros*

CMS/DOS supports the following imperative macros:

- *Initialization macros*: OPEN and OPENR
- *Processing macros*: GET, PUT, PUTR, RELSE, TRUNC, CNTRL, ERET, and PRTOV.

  **Note:** No code is generated for the CHNG macro.
- *Work file macros for tape and disk*: READ, WRITE, CHECK, NOTE, POINTR, POINTW, and POINTS.
- *Completion macros*: CLOSE and CLOSER

CMS/DOS supports workfiles containing fixed-length unblocked records and undefined records. Disk work files are supported as single volume, single pack files. Normal extents and split extents are both supported.

## VSE Transient Routines

CMS/DOS simulates the VSE transients that are fetched by macro expansion or by the LIOCS modules. These simulation routines contain enough of the transient's function to support the DOS/VS COBOL compiler and DOS PL/I Optimizing compiler. These routines that simulate the VSE transients execute in the CMS/DOS discontiguous shared segment.

The following VSE transients are simulated by CMS/DOS.

| Transient | Function under CMS/DOS |
|---|---|
| $$BOPEN | Fetched by the VSE OPEN macro expansion or by the VSE LIOCS modules. $$BOPEN performs DTF initialization, dependent upon the device type, to ready the file for I/O operations. At entry to $$BOPEN, register 0 points to a list of fullword addresses containing a pointer to the DTFs. $$BOPEN checks for supported DTF types, and initializes DTFs in accordance with the device type. In the case of tape data files, default DLBLs with the NOCHANGE option are issued. (The CMS STATE command is issued to verify the existence of the input files on disk.)<br><br>If a VSAM file is being opened (Byte 20 = X'28' in the ACB), control is passed to the VSAM OPEN routine. When opening DTFSD files for output or DTFCP/DTFDI disk files for output, if a file exists on a CMS disk with the same filename, filetype, and filemode, the file is erased. If a SAM disk file is being opened, DTF initialization is performed by involving the simulated VSE OPEN routines that reside in the CMSBAM DCSS. |
| $$BOPNLB | Fetched by COBOL Compiler Phase 00 to read the appropriate system or private source statement library directory record and to determine whether or not active members are present for the library. |
| $$BCLOSE | Fetched by VSE CLOSE macro expansion to deactivate a file. |
| $$BDUMP | Fetched when an abnormal termination condition is encountered. Control is not passed to a STXIT routine. CMS/DOS performs a CP dump to a virtual printer. The routine is canceled. |
| $$BOPENR | Fetched by a VSE OPENR macro expansion. The function of $$BOPENR is to relocate all DTF table address constants from the assembled addresses to executable storage addresses. At entry to $$BOPENR, register 0 points to an assembled address constant followed by a list of DTF addresses tables that require address modification. |
| $$BOPNR3 | Fetched by $$BOPENR to relocate all DTF table address constants for unit record DTFs. |
| $$BOPNR2 | Fetched by $$BOPNR3 to relocate all DTF table address constants for DTFDI or DTFCP. |
| $$BOSVLT | Fetched via SVC 2 by the simulated VSE OPEN/CLOSE routines in the CMSBAM DCSS. $$BOSVLT performs clean-up and transition functions upon completion of processing by the simulated VSE routines in the CMSBAM DCSS. |

## EXCP Support in CMS/DOS

CMS/DOS simulates the EXCP (execute channel program) routines to the extent necessary to support the LIOCS routines described in the preceding section, "VSE Supervisor and I/O Macros Supported by CMS/DOS."

Because CMS/DOS uses the VSE LIOCS routines, it must simulate all I/O at the EXCP level. The EXCP simulation routines convert all the I/O that is in the CCW

format to CMS physical I/O requests. That is, CMS macros (such as RDBUF/WRBUF, CARDRD/CARDPH, PRINTIO, and WAITRD/TYPLIN) replace the CCW strings. If CMS/DOS is reading from DOS disks, I/O requests are handled via the DIAGNOSE interface.

When an I/O operation completes, CMS/DOS posts the CCB or IORB with the CMS return code. Partial RPS (rotational position sensing) support is available for I/O operations to CMS disks because CMS uses RPS in its channel programs. However, RPS is not supported when real DOS disks are read.

## VSE Supervisor Control Blocks Simulated by CMS/DOS

CMS/DOS supports VSE program development and execution for a single partition: the background partition. Because CMS/DOS does not support foreground partitions, it also does not simulate the associated control blocks and fields for foreground partitions. CMS/DOS does simulate the following VSE supervisor control blocks:

- ABTAB--Abnormal Termination Option Table
- BBOX--Boundary Box
- BGCOM--Background Partition Communication Region
- EXCPW--Work area for module DMSXCP
- FICL--First in Class
- LUB--Logical Unit Block
- NICL--Next in Class
- PCTAB--Program Check Option Table
- PIBTAB--Program Information Table
- PIB2TAB--Program Information Block Table Extension
- PUB--Physical Unit Block
- PUBOWNER--Physical Unit Block Ownership Table
- SYSCOM--System Communication Region
- TCB--Task Control Block
- LOCTAB--LOCK/UNLOCK Resource Table
- DIB--Disk Information Block

For detailed descriptions of CMS/DOS control blocks, refer to the *VM/SP Data Areas and Control Block Logic, Volume 2*.

## User Considerations and Responsibilities

A critical design assumption of CMS/DOS is that installations that use CMS/DOS for VSE program development also use and have available a VSE system. Therefore, if you want to use CMS/DOS for VSE program development, you should order and install a VSE system. Also, if you want to use the DOS/VS COBOL and DOS PL/I Optimizing compilers under CMS/DOS, you must order them and install them on your VSE system.

You should consider several other facts if you plan to use CMS/DOS. The following sections describe some of the user considerations and responsibilities.

## VSE System Generation and Updating Considerations

The CMS/DOS support in CMS may use a real VSE system pack. CMS/DOS provides the necessary path and then fetches VSE logical transients and system routines directly as well as the DOS/VS COBOL and DOS PL/I Optimizing compilers directly from the VSE system or private core image libraries.

It is your responsibility to order a VSE system and then generate it. Also, if you plan to use DOS compilers, you must order the current level of the DOS/VS COBOL compiler and DOS PL/I Optimizing compiler and install them on the same VSE system.

When you install the compilers on the VSE system, you must link-edit all the compiler modules as relocatable phases using the following linkage editor control statement:

```
ACTION REL
```

You can place the link-edited phases in either the system or the private core image library.

When you later invoke the compilers from CMS/DOS, the library (system or private) containing the compiler phases must be identified to CMS. You identify all the system libraries to CMS by coding the filemode letter that corresponds to that VSE system disk on the SET DOS ON command when you invoke the CMS/DOS environment. You identify a private library by coding ASSGN and DLBL commands that describe it. The VSE system and private disks must be linked to your virtual machine and accessed before you issue the commands to identify them for CMS.

CMS/DOS has no effect on the update procedures for VSE, COBOL, or DOS PL/I. Normal update procedures for applying IBM-distributed coding changes apply.

For detailed information on how to generate VM/SP with CMS/DOS, refer to the publication *VM/SP Planning Guide and Reference* and the *VM/SP Installation Guide.*

## VM/SP Directory Entries

The VSE system and private libraries are accessed in read-only mode under CMS/DOS. If more than one CMS virtual machine is using the CMS/DOS environments you should update the VM/SP directory entries so that the VSE system residence volume and the VSE private libraries are shared by all the CMS/DOS users.

The VM/SP directory entry for one of the CMS virtual machines should contain the MDISK statements defining the VSE volumes. The VM/SP directory entries for the other CMS/DOS users should contain LINK statements.

For example, assume the VSE system libraries are on cylinders 0 through 149 of a 3330 volume labeled DOSRES. And, assume the VSE private libraries are on cylinders 0 through 99 of a 2314 volume labeled DOSPRI. Then, one CMS machine (for example, DOSUSER1) would have the MDISK statements in its directory entry.

```
USER DOSUSER1 password 320K 2M G
     .
     .
     .
MDISK 331 3330 0 150 DOSRES R rpass
MDISK 231 2314 0 100 DOSPRI R rpass
```

All the other CMS/DOS users would have links to these disks. For example

```
LINK DOSUSER1 331 331 R rpass
LINK DOSUSER1 231 231 R rpass
```

## When the VSE System Must Be Online

Most of what you do in the CMS/DOS environment for VSE program development requires that the VSE system pack and/or the VSE private libraries be available to CMS/DOS. In general, you need these VSE volumes whenever:

- You use the DOS/VS COBOL compiler or DOS/PLI Optimizing compiler. The compilers are executed from the system or private core image libraries.
- Your source programs contain COPY, LIBRARY, %INCLUDE, or CBL statements. These statements copy books from your system or the private source statement library.
- You invoke one of the library programs: DSERV, RSERV, SSERV, PSERV, or ESERV.
- You execute VSE programs that use LIOCS modules. CMS/DOS fetches most of the LIOCS routines for non-disk files directly from VSE system or private libraries.

A VSE system pack is usable when it is:

- Defined for your virtual machine
- Accessed
- Specified, by mode letter, on the SET DOS ON command.

A VSE private library is usable when it is:

- Defined for your virtual machine
- Accessed
- Identified via ASSGN and DLBL commands.

## Performance

Although you can use the CMS/DOS library services to place the DOS/VS COBOL compiler, DOS PL/I compiler, and ESERV program in a CMS DOSLIB, it is recommended that you do not use this method with 800-byte format CMS disks. CMS/DOS can fetch these directly from the VSE system or private libraries faster than from a DOSLIB on 800-byte format CMS disks. Fetch time from DOSLIBs on 512, 1K-, 2K-, or 4K-byte format CMS disks is approximately equivalent to that of VSE system or private libraries.

## Execution Considerations and Restrictions

The CMS/DOS environment does not support the execution of VSE programs that use:

- Teleprocessing or indexed sequential (ISAM) access methods. CMS/DOS supports only the sequential (SAM) and virtual storage (VSAM) access methods.
- Multitasking. CMS/DOS supports only a single partition, the background partition.

CMS/DOS can be executed in a CMS Batch Facility virtual machine. If any of the VSE programs that are executed in the batch machine read data from the card reader, you must ensure that the end-of-data indication is recognized. Be sure that (1) the program checks for end of data and (2) a /* record follows the last data record.

If there is an error in the way you handle end of data, the VSE program could read the entire batch input stream as its own data. The result is that jobs sent to the batch machine are never executed and the VSE program reads records that are not part of its input file.

# CMS Support for OS and VSE/VSAM Functions

CMS supports interactive program development for OS and VSE programs using VSAM. CMS supports VSAM macros for OS and VSE programs. The complete set of VSE/VSAM macros and options and a subset of OS/VSAM macros are supported.

CMS also supports Access Method Services to manipulate OS and VSE VSAM and SAM data sets.

Under CMS, VSAM data sets can span up to 10 volumes. CMS does not support VSAM data set sharing; however, CMS already supports the sharing of minidisks or full pack minidisks.

VSAM data sets created in CMS are not in the CMS file format. Therefore, CMS commands currently used to manipulate CMS files cannot be used for VSAM data sets that are read or written in CMS. A VSAM data set created in CMS (using VSE/VSAM) has a file format that is compatible with OS VSAM data sets as long as the physical record size of the data set is .5K, 1K, 2K, or 4K. For complete information on OS/VS VSAM and VSE/VSAM data set compatibility, see the *VSE/VSAM General Information Manual.*

Because VSAM data sets in CMS are not a part of the CMS file system, CMS file size, record length, and minidisk size restrictions do not apply. The VSAM data sets are manipulated with Access Method Services programs executed under CMS, instead of with the CMS file system commands. Also, all VSAM minidisks and full packs used in CMS must be initialized with the Device Support Facility; the CMS FORMAT command must not be used.

CMS supports VSAM control blocks with the GENCB, MODCB, TESTCB, and SHOWCB macros.

In its support of VSAM data sets, CMS uses RPS (rotational position sensing) wherever possible. CMS does not use RPS for 2314/2319 devices, or for 3340 devices that do not have the feature.

## Hardware Devices Supported

CMS support of VSAM data sets is based on VSE/VSAM. Except for the 3380, only disks supported by VSE can be used for VSAM data sets in CMS. These disks are:

- IBM 2314 Direct Access Storage Facility
- IBM 2319 Disk Storage
- IBM 3310 Direct Access Storage
- IBM 3330 Disk Storage, Models 1 and 2
- IBM 3330 Disk Storage, Model 11
- IBM 3340 Direct Access Storage Facility
- IBM 3344 Direct Access Storage
- IBM 3350 Direct Access Storage
- IBM 3370 Direct Access Storage
- IBM 3375 Direct Access Storage
- IBM 3380 Direct Access Storage (OS/VSAM environment of CMS only).

CMS disk files used as input to or output from Access Method Services may reside on any disk supported by CMS.

## VSE Supervisor Macros and Logical Transients Support for VSAM

CMS supports VSAM for OS and VSE users. However, the CMS support of VSAM is based on VSE/VSAM. VSE supervisor macros required by VSE/VSAM are supported by CMS. See Figure 45 on page 392 for a complete list of supervisor macros supported.

CMS distributes the VSE transients that are needed in the VSAM support. Thus, OS users do not need to have the VSE system pack online when they are compiling and executing VSAM programs.

CMS uses all of the VSE B-transients except those that build and release extent blocks. The extent block is not supported in CMS and, thus, neither are the B-transients that control extent blocks.

The CMSDOS shared segment contains the B-transients that are simulated for VSE support in CMS. The B-transients that pertain only to VSAM are included in the VSAM saved segment. Other VSE routines required by VSE/VSAM are contained in the CMSBAM shared segment. This includes the common VTOC handler routines, SAM data management, and the VSAM look-aside function.

## Data Set Compatibility Considerations

CMS can read and update VSAM data sets that were created under VSE or OS/VS. In addition, VSAM data sets created under CMS can be read and updated by VSE or OS/VS as long as the physical record size of the data set is .5K, 1K, 2K, or 4K.

However, if you perform allocation on a minidisk in CMS, you cannot use that minidisk in an OS virtual machine in any manner that causes further allocation. VSE/VSAM (and, thus, CMS) ignores the format-5, free space, DSCB, on VSAM disks when it allocates extents. If allocation later occurs in an OS machine, OS attempts to create a format-5 DSCB. However, the format-5 DSCB created by OS does not correctly reflect the free space on the minidisk. In CMS, allocation occurs whenever data spaces or unique data sets are defined. Space is released whenever data spaces, catalogs, and unique data spaces are deleted.

For complete information on OS/VS VSAM and VSE/VSAM data set compatibility, see the *VSE/VSAM General Information*.

## ISAM Interface Program (IIP)

CMS does not support the VSAM ISAM Interface Program (IIP). Thus, any program that creates and accesses ISAM (indexed sequential access method) data sets cannot be used to access VSAM key sequential data sets. There is one exception to this restriction. If you have (1) OS PL/I programs that have files declared as ENV(INDEXED) and (2) if the library routines detect that the data set being accessed is a VSAM data set, your programs will execute VSAM I/O requests.

# Saving the CMS System

Only named systems can be saved. The NAMESYS macro must be used to name a system. A discussion on creating a named system is found under "Generating Saved Systems" in "Part 1: Control Program (CP)."

The DMKSNT module must have been configured (by coding the NAMESYS macro) when CP was generated. The DMKSNT module contains the system name, size of the system, and its real disk location. The CMS system may be saved by entering the command "SAVESYS name" as the first command after the IPL command (that is, after the CMS version identification is displayed), where "name" is the name to be assigned to the saved system.

The CMS S and Y-disks (if the Y-disk is defined) must be mounted and attached to the virtual machine, creating the saved system before the SAVESYS command is issued. This ensures that CMS file directories are saved correctly.

Any updates to the CMS S-disk or Y-disk requires resaving the CMS system.

The IPLing of the saved CMS system is similar to IPLing by device except that the directories for the S and Y-disk are part of the nucleus instead of being built in DMSFREE storage.

In VM/SP, the CMS system is designed to be used as a saved system. Its location may be modified by an installation for its particular requirements, but should be shared among CMS users.

## Saved System Restrictions for CMS

There are several coding restrictions that must be imposed on CMS if it is to run as a saved system.

If the key specified in the CAW for a SIO instruction is zero, then the data area for input may not cross the boundary between two pages with different storage keys.

If you intend to modify a shared CMS system, be sure that all code that is to be shared resides in the shared segments of the CMS Nucleus (suggested location: X'1D0000' to X'200000' ). You can use the USERSECT area of DMSNUC to contain nonshared instructions.

CP does not permit a user of a shared system to set storage keys via the Set Storage Key (SSK) instruction. Thus, one user cannot prevent other users from accessing shared storage.

# The CMS Batch Facility

The CMS Batch Facility is a VM/SP programming facility that runs under the CMS subsystem. It allows VM/SP users to run their jobs in batch mode by sending jobs either from their virtual machines or through the real (system) card reader to a virtual machine dedicated to running batch jobs. The CMS Batch Facility then executes these jobs, freeing user machines for other uses.

If both CMS Batch Facility and the Remote Spooling Communications Subsystem (RSCS) are being executed under the same VM/SP system, job input streams can be transmitted to the batch facility from remote stations via communication lines. Also, the output of the batch processing can be transmitted back to the remote station.

The CMS Batch Facility virtual machine is generated and controlled on a userid dedicated to execution of jobs in batch mode. The system operator generates the "batch machine" by loading (via IPL) the CMS subsystem, and then issuing the CMSBATCH command. The CMSBATCH module loads the DMSBTP TEXT S2 file, which is the actual batch processor. After each job is executed, the batch facility IPLs itself, thereby providing a continuously processing batch machine. The batch processor IPLs itself by using the PARM option of the CP IPL command, followed by a character string that CMS recognizes as peculiar to a batch virtual machine performing its IPL. Jobs are sent to the batch machine's virtual card reader from users' terminals and executed sequentially. When there are no jobs waiting for execution, the CMS Batch Facility remains in a wait state ready to execute a user job. See the *VM/SP Operator's Guide* for more information about controlling the batch machine.

The CMS Batch Facility is particularly useful for compute-bound jobs such as assemblies and compilations and for execution of large user programs, since interactive users can continue working at their terminals while their time-consuming jobs are run in another virtual machine.

The system programmer controls the batch facility virtual machine environment by resetting the CMS Batch Facility machine's system limits, by writing routines that handle special installation input to the batch facility, and by writing EXEC procedures that make the CMS Batch Facility facility easier to use.

## Installing the CMS Batch Machine

Before using the CMS Batch Facility, an entry must exist in the users directory. This entry specifies the userid of the CMS Batch machine.

Following is an example of a user directory entry granting authorization to use the CMS Batch Facility.

```
USER CMSBATCH BATCH 1M 2M BG
     ACCOUNT 13 SYSTEM
     OPTION ACCT
     IPL CMS
     CONSOLE 009 3215
     SPOOL 00C 2540 READER *
     SPOOL 00D 2540 PUNCH A
     SPOOL 00E 1403 A
     LINK MAINT 190 190 RR
     MDISK 195 3330 xxx 010 'paswrd' W 'rdpswd' 'wrtpswd' 'allpswd'
```

Consult *VM/SP Planning Guide and Reference* the proper coding of the directory macro parameters.

**Note:** There is no 191 MDISK for the CMS Batch Machine.

In order to have the CMS Batch Machine autologed, you should have the following entry in the autolog virtual machine's PROFILE EXEC:

```
AUTOLOG CMSBATCH BATCH CMSBATCH
```

Otherwise, the operator logs on to the CMS Batch Machine and enters "CMSBATCH" followed by "DISCONNECT" (if the CMS Batch Machine is to run in DISCONNECT status).

**Note:** Refer to *VM/SP CP Command Reference for General User's* for more information on the AUTOLOG command.

## Resetting the CMS Batch Facility System Limits

Each job running under the CMS Batch Facility is limited by default to the maximum value of 32,767 seconds of virtual processor time, 32,767 punched cards output, and 32,767 printed lines of output. You can reset these limits by modifying the BATLIMIT MACRO file, which is found in the CMSLIB macro library, and by reassembling DMSBTP.

## Writing Routines To Handle Special Installation Input

The CMS Batch Facility can handle user-specified control language and special installation batch facility /JOB control cards. These handling mechanisms are built into the system in the form of user exits from batch; you are responsible for generating two routines to make use of them. These routines must be named BATEXIT1 and BATEXIT2, respectively, and must have a filetype of TEXT and a filemode number of 2 if placed on the system disk or an extension of the system disk. (See the *VM/SP CMS User's Guide* for information on how to write and use CMS Batch Facility control cards.) The routines you write are responsible for saving registers, including general register 12, which saves addressability for the batch facility. These routines (if made available on the system disk) are included with the CMS Batch Facility each time it is loaded.

### *BATEXIT1: Processing User-Specified Control Language*

BATEXIT1 is an entry point provided so that users may write their own routine to check non-CMS control statements. For example, a routine could be written to scan for the OS job control language needed to compile, link edit, and execute a FORTRAN job. BATEXIT1 receives control after each read from the CMS Batch Facility virtual card reader is issued. General register 1 contains the address of the

batch facility read buffer, which contains the card image to be executed by the batch facility. This enables BATEXIT1 to scan each card it receives as input for the type of control information you specify.

If, after the card is processed by BATEXIT1, general register 15 contains a nonzero return code, the CMS Batch Facility flushes the card and reads the next card. If a zero is returned in general register 15, the batch facility continues processing by passing the card to CMS for execution.

### *BATEXIT2: Processing the Batch Facility /JOB Control Card*

BATEXIT2 is an entry point provided so that users can code their own routine to use the /JOB card for additional information. BATEXIT2 receives control before the VM/SP routine used to process the batch facility /JOB card begins its processing, but after CMS has scanned the /JOB card and built the parameter list. When BATEXIT2 is processing, general register 1 points to the CMS parameter list buffer. This buffer is a series of 8-byte entries, one for each item on the /JOB card. If the return code found in general register 15 resulting from BATEXIT2 processing of this card is nonzero, an error message is generated and the job is flushed. If general register 15 contains a zero, normal checking is done for a valid userid and the existence of an account number. Finally, execution of this job begins.

## EXEC Procedures for the Batch Facility Virtual Machine

You can control the CMS Batch Facility virtual machine using EXEC procedures. For example, you can use an EXEC:

- To produce the proper sequence of CP/CMS commands for users who do not know CMS commands and controls.
- To provide the sequence of commands needed to execute the most common jobs (assemblies and compilations) in a particular installation.

For information on how to use the EXEC facility to control the batch facility virtual machine, see the *VM/SP CMS User's Guide.*

## Data Security under the Batch Facility

After each job, the CMS Batch Facility loads (via IPL) itself, destroying all nucleus data and work areas. All disks to which links were established during the previous job are detached.

At the beginning of each job, the batch facility work disk is accessed and then immediately erased, preventing the current user job from accessing files that might remain from the previous job. Because of this, execution of the PROFILE EXEC is disabled for the CMS Batch Facility machine. You may, however, create an EXEC procedure called BATPROF EXEC and store it on any system disk to be used instead of the ordinary PROFILE EXEC. The batch facility then executes this EXEC at each job initialization time.

## Improved IPL Performance Using a Saved System

Since the CMS Batch processor goes through an IPL procedure after each user job, an installation may experience a more efficient IPL procedure by using a saved CMS system when processing batch jobs.

This can be accomplished by passing the name of the saved system to the CMS Batch Facility via the optional "sysname" operand in the CMSBATCH command line.

The batch facility saves the name of the saved system until the end of the first job, at which time it stores the name in the IPL command line both as the "device address" and as the PARM character string. The latter entry informs the CMS initialization routine (DMSINS) that a saved system has been loaded and that the name is to be saved for subsequent IPL procedures.

**Note:** When using the CMS SET command, the BLIP operand is ignored when issued from the CMS batch machine.

# The Programmable Operator Facility

## Overview

The Programmable Operator Facility is designed to increase the efficiency of system operation and to allow remote operation of systems in a distributed data processing environment. It does this by intercepting all messages/requests directed to its virtual machine and by handling them according to preprogrammed actions. It determines whether a message is to be simply recorded for future reference, whether the message is to be acted upon, or whether the message is to be sent on to the operator to handle.

### Use in a Single System

When the programmable operator facility is operational in a single-system environment, it can:

- Ease message traffic to the system operator, by:
  - Filtering (logging) non-essential, information-only messages
  - Routing messages (for example, I/O intervention requests) to someone else for specialized action.
- Increase productivity, by freeing the system operator from certain **routine** responses or tasks. Such responses (whether they consist of one or a series of commands, whether VM/SP or guest operating system) may be preprogrammed to execute automatically upon receipt of a given message.

  Thus, only essential, non-routine messages (that is, those requiring the skill and experience of a system operator to handle) are sent on to the operator for response or action.

### Use in Distributed Systems

The capabilities of the programmable operator, outlined above, also allow for the remote operation of systems in a distributed environment. When the programmable operator facility is operational in a distributed system, it can:

- Issue responses and perform tasks that do not require an on-site operator
- Filter (log) non-essential, information-only messages
- Route messages requiring on-site (that is, manual) intervention to someone, not necessarily an operator, at the distributed site for action
- Route messages that require the skill and experience of a system operator to handle to the operator at the host system. The operator at the host site can also send commands to the programmable operator facility to control its operation, as well as commands to execute on the distributed system to control the system itself.

By running the programmable operator facility on VM/SP systems distributed at several different locations (network nodes), one operator at a host site can control a network of systems.

## The Logical Operator

Occasionally the programmable operator must send messages to another virtual machine. To ensure that the programmable operator will function properly, a user (a virtual machine other than the programmable operator virtual machine on the local system or in a distributed system) is identified to the programmable operator to receive these messages. This user is called the **logical operator**, as opposed to the CP system operator. When the programmable operator is started (in the CP system operator virtual machine, for example), the logical operator virtual machine receives an initiation message. The logical operator also receives error messages for severe errors, such as logging errors, and receives all messages routed to the logical operator explicitly or by default.

## How it Works

The programmable operator facility runs in a CMS virtual machine. Although it can run in any virtual machine, because of its programmed capability to log, handle, or redirect messages, it is most commonly run in the CP system operator's virtual machine.

The programmable operator facility compares all messages directed to it against entries listed in a routing table (a CMS file). When a match occurs, the prescribed action is performed. Any messages that require a real operator's response or action are sent on to the defined operator (system, network, etc.) at another virtual machine console, a "logical" operator's console. If the logical operator's virtual machine is in the same system, the programmable operator sends the messages with either the CP MESSAGE or CP MSGNOH command. If the logical operator's virtual machine is in a different system (network node), a host system for example, it sends the messages via RSCS Networking.

Consider this example:
The SYSOPR macro in DMKSYS specifies the userid OPER1 for the CP system operator. Set up the programmable operator to run in the OPER1 virtual machine and establish another virtual machine with userid OPERX. In the routing table file(s), specify OPERX as the logical operator. Now any CP or user messages sent to the system operator virtual machine can be handled or filtered by the programmable operator or routed to userid OPERX.

## Flow of Operation

When the programmable operator facility is running in a virtual machine, CP intercepts all messages intended for that virtual machine console. CP then passes these messages to the programmable operator facility via IUCV. The messages are logged in a CMS file. The programmable operator facility then uses the active routing table to analyze the message and determine if further action is needed. Based on the contents of the routing table (such as message texts, message types, and user authorizations), the message can be passed to some specified action routine for further action. If the message is to be routed to the logical operator, and that person is on another virtual machine in the same physical machine, the programmable operator facility routes the message directly to the logical operator via the CP MSGNOH command or the CP MESSAGE command depending on the classification of the programmable operator virtual machine. If the logical operator is on a different physical machine, the programmable operator facility prefaces the message with the appropriate tag information and sends the message to RSCS Networking via the CP SMSG command.

The programmable operator facility usually operates in a disconnected virtual machine. If someone logs on to this disconnected virtual machine with the programmable operator facility running, no messages are displayed (unless the programmable operator facility is running in DEBUG mode). All messages are being intercepted or received by the programmable operator program from IUCV. If that person should enter a command, the programmable operator facility gets control and reads the command entered. Only two commands are accepted from this environment; the STOP command and the SET command. The programmable operator facility rejects any other commands.

If a CMS abend occurs while the programmable operator facility is executing, all files are closed and abend error messages are sent to the logical operator. A dump of the virtual machine storage is taken using the CP VMDUMP command and the last system or device that was IPLed is re-IPLed. If the abend occurs while an action routine is executing, abend error messages are sent to the logical operator and the requester (if any). Control is returned to the point in the programmable operator facility immediately following the action routine call.

### Relationship to RSCS Networking

When the programmable operator facility is running in a network environment, it is a normal user of RSCS Networking. This means that the programmable operator facility communicates to RSCS via the CP SMSG command. Any configuration of systems and networks that are supported by RSCS Networking can use the programmable operator facility. The time needed for a message to go from the system at a distributed site to the logical operator at the host system, or vice versa, depends on the number and type of communications links between the message sources and destinations.

A programmable operator can check on its ability to communicate with a host or distributed system. See "Communications Checking" later in this section.

### Routing Table Information

The programmable operator routing table identifies the programmable operator facility environment, including the logical operator's virtual machine id and nodeid. It also specifies the action to take for each message, and authorizes certain users to invoke specific programmable operator commands. For a complete description of the information contained in a routing table, see "The Routing Table" later in this section.

The routing table is a separate CMS file that must be tailored for a specific use. The first routing table to be used is specified when the programmable operator facility is invoked. If no routing table name is specified, the default filename "PROP" is used.

The installation may define multiple routing tables to cover varying situations. For example, multiple routing tables can be defined to cover shift changes. Only one routing table can be active at a time. The active routing table may be replaced by issuing the LOADTBL command. Any person authorized in the active routing table may issue this command.

### Initialization

The programmable operator facility is initiated by IPLing a CMS virtual machine of at least 512K in size and invoking the programmable operator facility. When the programmable operator facility gets control, it locates the specified or default rout-

ing table and loads it into virtual storage. For each action routine specified in the routing table, an EXEC file or a corresponding member in a CMS simulated OS load library named PROPLIB LOADLIB must exist. If an EXEC does not exist, the LOADLIB member is loaded as a nucleus extension via the NUCXLOAD command. If both exist, the EXEC takes precedence.

If upon invocation, the programmable operator facility cannot find an action routine named in the routing table, an error message is issued, and the programmable operator facility terminates operation. Otherwise, the programmable operator facility is fully initialized, and writes a message to the programmable operator's console, to the logical operator, and to the LOG file, indicating that the programmable operator facility has started. The programmable operator facility then waits for either an interrupt indicating an incoming message or an interrupt from the console.

**Note:** If the user enters a nodeid into the SYSTEM NETID file that is invalid as a CMS filetype, the programmable operator cannot start because it is not be able to open the log file.

## The Routing Table

The routing table is a CMS file that contains the information used to control the operation of the programmable operator facility. The routing table enables the programmable operator facility to recognize a message as a command, to determine the action to take when a message comes in, and to recognize the authorized users of programmable operator functions.

### *How the Programmable Operator Facility Uses the Routing Table*

When the programmable operator facility receives an IUCV interrupt with an incoming message, the active routing table is searched to find a matching entry. When the routing table is searched, all fields are checked. In order for a match to occur, each field must either match or be blank. If a matching entry is found, that entry contains information pertaining to any action to be taken. The action routine name tells the programmable operator facility which action routine to invoke when a routing table entry matches the incoming message. If no matching entry is found in the active routing table, no action is taken besides logging the message.

The order that the entries are placed in the routing table affects the way the programmable operator facility performs. The routing table is searched from top to bottom until a match is found. As the table is searched, lines that begin with an asterisk (*) in column 1 are ignored, and therefore may be used to place comments in the routing table. Also, lines that are completely blank are ignored in the routing table search and can be used to separate lines of text for easier reading. All entries must be made in upper case.

**Note:** The routing table format is changed from the initial version of the programmable operator facility in VM/SP Release 2. The original format from Release 2 is not compatible with later versions of the programmable operator facility. The routing tables must be converted to reflect this change. See "Routing Table Conversion" later in this section.

### *Routing Table Entry Formats*

Every routing table must have specific configuration information in the first records of the routing table file (filetype RTABLE) that are not comments or blank lines.

These statements are in free format, meaning that they need not be positioned in any particular columns. See Figure 53 on page 431 for an example of a partial routing table. The statements and their parameters are as follows:

1. The **LGLOPR** statement identifies the logical operator.

| | |
|---|---|
| LGLOPR | {userid [nodeid]}<br>{nickname } |

*where*:

userid is a valid userid on the specified node.

nodeid is a valid id of a system in the network. If no nodeid is specified, the local system's nodeid is used.

nickname is a nickname defined in the programmable operator facility virtual machine CMS NAMES file.

**Note:** If a nickname is used to identify the logical operator, the nickname cannot be a list of nicknames. The programmable operator must have one nodeid to associate with the logical operator.

Either a nickname or a userid must be specified. If a userid is specified, a nodeid may be specified. If both a userid and a nodeid are specified, they must be separated by one or more blanks. If the name specified is both a local userid and a nickname, the programmable operator regards it as a nickname.

**IMPORTANT NOTE:** The programmable operator virtual machine should not be identified as the logical operator. This causes the programmable operator to go into a loop in the event it tries to do the routing. This includes specifying a userid of OPERATOR (or any abbreviation thereof). This also causes the message to be sent to the system operator virtual machine, even if the system operator virtual machine has a different userid.

2. The optional **TEXTSYM** statement specifies the characters that the programmable operator facility interprets as special symbols in the text field of the routing table entries. All three parameters must be specified if the statement is specified.

| | |
|---|---|
| TEXTSYM | blank-sep arbchar-sep not-symbol |

*where*:

blank-sep is a separator character indicating that **blanks** are to be skipped over when scanning the message. A message is scanned for the next non-blank character string. This non-blank character string is then compared to the text in the routing table entry following this separator character. The default character is "/".

arbchar-sep is a separator character indicating that **all non-matching characters** are to be skipped over when scanning the message. A message is scanned for the text specified in the routing table entry until it is found or until the end of the message is reached. The default character is "$".

not-symbol — when it immediately follows a separator is the character indicating that the text should not be found in the message. If the text following the not-symbol is found in the message, then the message does not match that routing table entry. The default character is "¬".

See "Filtering Messages" for the use of TEXTSYM characters in routing table entries.

3. The optional **PROPCHK** statement identifies the distributed nodes that the host system is to check on. The RSCS nodeids of these distributed systems must be specified in this statement. A programmable operator must be running in the system operator virtual machine on the distributed systems being checked. As many nodes may be specified on one statement as fit in an 80-column record. The programmable operator facility only reads the first 80 columns. Any number of PROPCHK statements may be entered to specify different checking or response wait intervals for different RSCS nodes. The PROPCHK statement must be after the LGLOPR statement.

   **Note:** Nodes to be checked with PROPCHK must be systems running VM/SP Release 3, with a programmable operator in the system operator virtual machine.

| PROPCHK | ccc ww nodeid [nodeid ...] |
|---|---|

   *where*:

   ccc — is the checking interval. This interval, in minutes, indicates how often acknowledgment requests are sent out to the specified nodes.

   ww — is the response wait interval. This interval is the number of minutes permitted to pass before a response must be received from the specified node(s).

   nodeid — is a valid id of a system in the network.

   **Notes:**

   a. The checking interval specified must be greater than the response wait interval.

   b. The nodeid of the logical operator must not be specified as a nodeid on this statement.

4. The optional **HOSTCHK** statement specifies the time interval for checking communication with the RSCS virtual machine at the logical operator node and the wait time for a response. The HOSTCHK statement must be after the LGLOPR statement.

| HOSTCHK | ccc ww |
|---|---|

   *where*:

ccc is the checking interval. This interval, in minutes, indicates how often acknowledgment requests are sent out to the specified nodes.

ww is the response wait interval. This interval is the number of minutes permitted to pass before a response must be received from the specified node(s).

**Note:** The checking interval specified must be greater than the response wait interval.

5. The optional **LOGGING** statement specifies whether messages or messages and command responses are to be logged or not logged. If the LOGGING statement is not in the routing table, messages are logged and LOGGING is ON. If the LOGGING statement is in the routing table, one of the three operands must also be specified, because there is no default operand.

| | |
|---|---|
| LOGGING | {ON / ALL / OFF} |

*where*:

ON indicates that messages are to be logged while this RTABLE is active, unless it is explicitly turned off using the SET LOGGING command.

ALL indicates that messages and programmable operator command responses are to be logged while this RTABLE is active, unless it is explicitly turned off using the SET LOGGING command.

OFF indicates that messages are not to be logged while this RTABLE is active, unless it is explicitly turned on using the SET LOGGING command.

6. The **ROUTE** statement indicates the end of the configuration statements and the start of the routing entries.

| | |
|---|---|
| ROUTE | |

This statement **must** follow the other statements specified in this section.

The LGLOPR and ROUTE statements are required in every routing table. An example of these statements in a routing table is as follows:

```
LGLOPR OPERATNS HOSTNODE
TEXTSYM / $ ¬
PROPCHK 5 1 NODE1 NODE2 NODE3
PROPCHK 3 1 NODE4 NODE5
HOSTCHK 2 1
LOGGING ALL
ROUTE
```

These special statements may be specified for each routing table in any order (as long as LGLOPR is first and ROUTE is last) and with at least one blank separating each parameter. The statements depend on the installation and, therefore, must be

supplied by the installation for each routing table. These entries are processed only when the routing table is loaded, so they are not searched during programmable operator message handling.

The configuration shown in Figure 52 can be described in a routing table with the first few lines like those in Figure 53 on page 431.

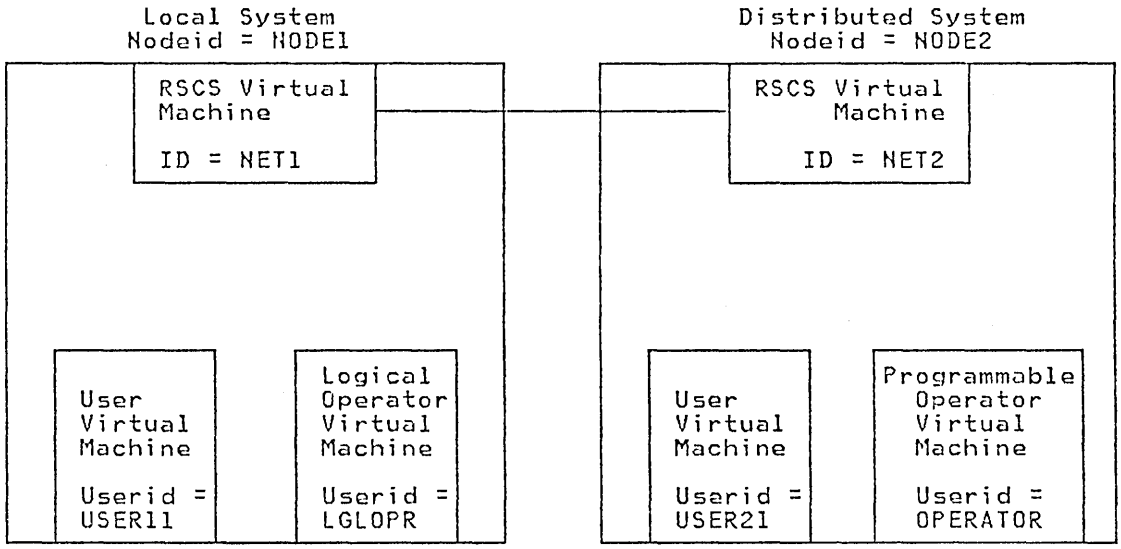



**Figure 52. The Programmable Operator Facility in a Distributed System.** The logical operator is situated at the Host system and the programmable operator is running in a different system at a distributed site.

The routing table entries to be searched must be in the following fixed format.

**Note:** The words in parentheses correspond to the vertically aligned words in the comment records in Figure 53 on page 431 through Figure 57, and also in the IBM sample routing table file.

| FIELD | EXAMPLE | FIELD COLUMNS | LENGTH OF FIELD |
|---|---|---|---|
| COMPARISON TEXT (TEXT) | /FEEDBACK / | 1-25 | 25 |
| STARTING COLUMN (SCOL) | 1 | 27-29 | 3 |
| ENDING COLUMN (ECOL) | 9 | 31-33 | 3 |
| IUCV MESSAGE CLASS (TYPE) | 1 | 35-36 | 2 |
| USERID (USER) | USER21 | 38-45 | 8 |
| NODEID (NODE) | NODE2 | 47-54 | 8 |
| ACTION ROUTINE NAME (ACTN) | DMSPOR | 56-63 | 8 |
| PARAMETER TO ACTION ROUTINE (PARM) | TOFB | 65-72 | 8 |

*where*:

COMPARISON TEXT
is a particular character string that the programmable operator facility is to search for in the incoming message. If this field is left blank, any text compared with this field is considered a match. Multiple texts may be specified, with the capability to skip over intervening blank or non-blank characters.

STARTING COLUMN
is the column in the incoming message where the programmable operator facility starts looking for the character string mentioned in the COMPARISON TEXT field. If this field is left blank, the programmable operator facility starts scanning at the beginning of the message.

ENDING COLUMN
is the column in the incoming message where the programmable operator facility stops looking for the character string(s) mentioned in the COMPARISON TEXT field. If this field is left blank, the programmable operator facility continues scanning until the end of the message.

IUCV MESSAGE CLASS
identifies the origin of the incoming message according to the IUCV message type. For more details on message types, see the Message System Service section of this manual. If this field is left blank, any value compared with this field is considered a match.

The IUCV message types are:

| Class | Message Types |
|---|---|
| 1 | Message sent using CP MESSAGE and CP MSGNOH. |
| 2 | Message sent using CP WARNING. |
| 3 | Asynchronous CP messages and CP responses to a CP command executed by the programmable operator facility virtual machine. |
| 4 | Message sent using CP SMSG command. |
| 5 | Any data directed to the virtual console by the virtual machine (WRTERM, LINEDIT, etc.). |
| 6 | Error messages from CP (EMSG). |
| 7 | Information messages from CP (IMSG). |
| 8 | Single Console Image Facility (SCIF) message from CP. |

**Note:** CP responses that are trapped in a buffer using the extended DIAGNOSE X'08' do not become type-3 messages. For example, CP responses from the programmable operator CMD command are not type 3-messages, and therefore are not logged when LOGGING is set to "ON".

USERID
is the character string that is compared to the VM userid of the user that sent the incoming message to the programmable operator facility. It determines the authority of the user to cause an action to be performed. If this field is left blank, all userids compared with this field are assumed to match.

NODEID
is the character string that is compared to the VM nodeid of the user that sent the incoming message to the programmable operator facility. Again, it determines the authority of the user to cause an action to be performed. If this field is left blank, all nodeids compared with this field are assumed to match.

ACTION ROUTINE NAME
is the name of the LOADLIB member (a Basic Assembler Language routine) that the programmable operator facility is to NUCXLOAD when the LOADTBL function is performed, or the name of an EXEC, and subsequently,

the routine that the programmable operator is to call when a match occurs on the entry in which the name is specified. If this field is left blank, no action is performed.

PARAMETER TO ACTION ROUTINE
is a character string of up to eight bytes that is passed as a parameter to the action routine by way of the programmable operator PARMLIST for a Basic Assembler Language routine or by way of a program stack for an EXEC. Often, this is used to specify a particular subroutine in the action routine. If this field is left blank, no parameter is passed.

Column 73 and beyond are reserved for future use.

```
* THIS IS THE DEFINITION OF THE PROP CONFIGURATION.
* LOGICAL OPERATOR IS NICKNAME "LOP".  SEE "OPERATOR NAMES" FILE.
* LOGICAL OPERATOR (NICKNAME "LOP") IS "OPERATOR" AT NODEID "NODE1".
LGLOPR   LOP
* THE TEXT SEPARATOR CHARACTERS.
TEXTSYM / $ ¬
* WHICH NODES TO CHECK, AND AT WHAT INTERVAL.
HOSTCHK  5  1
* THE ROUTING ENTRIES START
ROUTE
*------------------------ --- --- -- -------- -------- -------- --------
*T                        S   E   T  U        N        A        P
*E                        C   C   Y  S        O        C        A
*X                        O   O   P  E        D        T        R
*T                        L   L   E  R        E        N        M
*------------------------ --- --- -- -------- -------- -------- --------
* SEND PROP FEEDBACK COMMAND TO FEEDBACK ACTION ROUTINE
*------------------------ --- --- -- -------- -------- -------- --------
/FEEDBACK /                 1   9  1 USER21   NODE2    DMSPOR   TOFB
*------------------------ --- --- -- -------- -------- -------- --------
* FILTER OUT LOGON AND LOGOFF MESSAGES SO OPERATOR NEEDN'T SEE THEM
* BUT LET "FORCED" LOGOFF MESSAGES THROUGH
*------------------------ --- --- -- -------- -------- -------- --------
$LOGON                     18  24  3
$LOGOFF$¬FORCED            18      3
*------------------------ --- --- -- -------- -------- -------- --------
* FILTER OUT COMMANDS THAT WE DON'T WANT ISSUED.
*------------------------ --- --- -- -------- -------- -------- --------
/CMD/SYSTEM                                              WARNING
/CMD/SET/EC                                              WARNING
*------------------------ --- --- -- -------- -------- -------- --------
* ALLOW ONLY OPERATOR ON HOST TO ISSUE SHUTDOWN.
*------------------------ --- --- -- -------- -------- -------- --------
/CMD/SHUTDOWN                        OPERATOR NODE1    DMSPOR   TOVM
*------------------------ --- --- -- -------- -------- -------- --------
* ROUTE ALL MESSAGES ABOUT DEVICE 00E TO THE SPOOL OPERATOR.
*------------------------ --- --- -- -------- -------- -------- --------
$ 00E                          3                      DMSPOS   SPOOLOP
```

*where*:

"WARNING" represents a user action routine which may send a warning message to the user issuing that command.

"SPOOLOP" represents a nickname or userid of the spool operator.

**Figure 53. Partial routing table**

## *Tailoring the Routing Table*

Routing table entries determine what messages the programmable operator facility ignores (filtering), who is authorized to issue a particular command (authorization), and what action routines to invoke for a given circumstance. The routing table is tailored to suit an installation's individual needs by adding or changing entries in the routing table.

The programmable operator facility comes with a general purpose routing table named "PROP RTABLE". (See the section on "Installing the Programmable Operator Facility" to locate the "PROP RTABLE".) This supplied routing table can only be used after the LGLOPR, HOSTCHK, and PROPCHK statements are modified. The routing table entries may also have to be modified. These changes can be made by using the VM/SP System Product Editor. The programmable operator facility operates satisfactorily with no further changes. However, if an installation chooses, the supplied routing table can be modified to change the operation of the programmable operator facility. Different routing tables can also be created to cover varying circumstances. These tables can be dynamically loaded using the LOADTBL command. Only one routing table may be active at a time.

### Specifying Routing Texts

Here are more examples of text comparisons for the programmable operator facility. "$" is the arbitrary character separator, "/" is the blank separator, and "¬" is the not-symbol. In all of these examples, it is assumed that the starting and ending columns do not interfere with the matching.

1. The RTABLE entry

   ```
   $LOGOFF
   ```

   is matched by any message containing the word "LOGOFF". If one text is preceded by the arbitrary character separator ($), the text can appear anywhere in the message to be a match.

2. The RTABLE entry

   ```
   /LOGOFF
   ```

   matches the message

   ```
   LOGOFF USER1 IN 5 minutes
   ```

   as there are no non-blank characters preceding the word "LOGOFF", but the entry does not match the message

   ```
   11:20:15 GRAF 0A0 LOGOFF AS USER1      USERS = 020
   ```

   If only one text is preceded by the blank character separator (/), the text must be the first non-blank string in the message order to be a match.

3. The RTABLE entry

   ```
   $AUTO$LOGON$AUTOLOG
   ```

   matches the message

```
11:09:02 AUTO LOGON    ***    USER2      USERS = 021 BY AUTOLOG1
```

and the message

```
11:09:02 AUTO LOGON    ***    AUTOLOG2   USERS = 021 BY SYSTEM
```

but not the message

```
11:09:02 GRAF 0A0 LOGON AS AUTOLOG1    USERS = 023
```

or the message

```
AUTOLOG WON'T LOGON TOMORROW
```

A text with two or more texts preceded by arbitrary character separators ($), is matched by a message with all those texts appearing in that order.

The texts in the message are scanned in the order that they appear in the routing table entry. One text is searched for at a time. If the arbitrary character separator ($) precedes the text in the entry, a message is scanned until a match is found or the end of the message is reached. If the blank character separator (/) precedes the comparison text, blanks are skipped over and the first non-blank string of characters is compared to the comparison text, which may or may not match.

Routing table entries and messages are also affected if the text is preceded by a not-symbol (¬). The not symbol is always used with one of the other separator characters; it never stands alone. If a matching text is found and the text in the routing table is preceded only by a "$" or a "/", the position following the last matched text is remembered. If there are no more RTABLE texts to be searched for, the entry is a match. If there is another text in that RTABLE entry to be searched for, the scan continues from the position following the last matched text. A match depends on the rest of the message text and the routing table entry. If a matching text is found but the text in the routing table *is* preceded by the not-symbol (¬), the entry *is not* a match and checking goes no further. Similarly, if a matching text *is not* found but the text in the routing table *is not* preceded by the not-symbol, the entry *is not* a match. If a match *is not* found and the text *is* preceded by the not-symbol (¬), and if there are no more texts, the entry matches the message. If there are more texts to scan for, the scan continues as above starting with the character following the last match. A match depends on the rest of the message text and the routing table entry.

Consider the following example:

4. The RTABLE entry

   ```
   $¬AUTO$LOGON
   ```

   does match the message

   ```
   12:04:28 GRAF 0A0 LOGON AS USER1       USERS = 027
   ```

   Because the first text in the RTABLE entry (AUTO) is preceded by the arbitrary character separator ($), the entire text is searched for "AUTO". No match is found. Because the text is preceded by the not-symbol (¬), the text is still a match at this point. The scan for the next text (LOGON) begins at the end of the last match. Because there was no previous match, the scan begins again at the start of the message. The LOGON text is preceded by the arbi-

trary character separator ($), so the search proceeds through the message until "LOGON" is matched. Because "LOGON" appears in the message, this RTABLE entry and message *do* match.

Now consider this example:

5. The RTABLE entry

   ```
   $¬AUTO/LOGON
   ```

   does not match the same message

   ```
   12:04:28 GRAF 0A0 LOGON AS USER1       USERS = 027
   ```

   The message is scanned for "AUTO" as above. The search for "LOGON" again begins at the beginning of the message. In this case, however, the LOGON text is preceded by the blank separator (/), so only blanks are skipped prior to the comparison. No blanks are found, so the comparison is made at the beginning of the text and "LOGON" is compared with "12:04:". This is not a match. Because this text was not preceded by the not-symbol, this RTABLE entry and message *do not* match.

Another example:

6. The RTABLE entry

   ```
   $¬AUTO/LOGON
   ```

   does not match the message

   ```
   12:04:28 AUTO LOGON *** USER1          USERS = 027 BY AUTOLOG1
   ```

   Because "AUTO" is found in the message and is preceded in the RTABLE entry by the arbitrary character separator ($) and the not-symbol (¬), the RTABLE entry and message *do not* match.

Here is another example:

7. The RTABLE entry

   ```
   $LOGOFF$¬030/FORCED
   ```

   does not match the message

   ```
   12:04:28 USER DSC LOGOFF AS USER1      USERS = 026 FORCED
   ```

   The first text, "LOGOFF", is preceded by the arbitrary character separator ($) and is scanned for through the text. "LOGOFF" is found. Because "LOGOFF" is not preceded by the not-symbol, the next text is scanned. The scan continues from the end of the previous match, which is the character following the LOGOFF text. Since the arbitrary character separator ($), precedes "030", the entire remaining text is searched for "030". It is not found but because "030" is preceded by the not-symbol, the message and RTABLE entry still match. Finally, "FORCED" is scanned for. It is preceded by the blank separator (/). Blanks are skipped, and starting with the character following the last matched string (which was "LOGOFF"), "FORCED" is com-

pared to "AS USE". This is not a match. Because "FORCED" is not matched and is not preceded by the not-symbol, this RTABLE entry and message *do not* match.

Here are routing table entries that *do* match this message:

```
$LOGOFF$¬030$FORCED
```

would match because arbitrary characters would be skipped before comparison for "FORCED" and

```
$LOGOFF$¬030/¬FORCED
```

would match because the first nonblank string after "LOGOFF" is not "FORCED".

### Filtering Messages

This is the simplest application of the programmable operator facility. Entries can be placed in the routing table to filter informational messages. The messages are filtered because no action routine is specified in the routing table entries. For example, when the programmable operator facility is running in the system operator's virtual machine, informational messages resulting from commands such as, LOGON, LOGOFF, and DISCONN, can be prevented from being displayed at the logical operator's console. Although the messages are not displayed at the operator's console, they can be logged in the current day's log file. The routing table entries must identify the text(s) in the message that makes it unique and identify the columns between which the text(s) should be found in the message. With single or multiple texts, TEXTSYM characters should be selected accordingly. Figure 54 shows an example of how entries may be placed in the routing table to filter unwanted responses directed to the logical operator. For example, using Figure 54 below as the routing table, the message

```
12:04:50 GRAF 055 LOGON AS USER1
```

would match the second routing table entry (/¬AUTO$LOGON). This RTABLE specification means that, starting in column 9 (SCOL) of the message, "AUTO" cannot be the first non-blank string and that "LOGON" must appear somewhere in the message. The message would be filtered out but logged in the current day's log file.

However, the message

```
12:04:28 AUTO LOGON *** USER BY AUTOLOG1
```

would not match the second routing table entry (/¬AUTO$LOGON) because "AUTO" is the first non-blank string in the message appearing in the columns between the SCOL and ECOL fields. Thus, the message would not be filtered out and would be routed to the logical operator, as specified in the last entry in Figure 54.

```
* THIS IS THE DEFINITION OF THE PROP CONFIGURATION.
* LOGICAL OPERATOR IS NICKNAME "LOP".  SEE "OPERATOR NAMES" FILE.
LGLOPR   LOP
* THE TEXT SEPARATOR CHARACTERS.
TEXTSYM / $ ¬
* WHICH NODES TO CHECK, AND AT WHAT INTERVAL.
PROPCHK  5  1  NODE2A    NODE2B
PROPCHK  2  1  NODE1A    NODE1B
* THE ROUTING ENTRIES START
ROUTE
*----------------------- --- --- -- -------- -------- -------- --------
*T                        S   E   T  U        N        A        P
*E                        C   C   Y  S        O        C        A
*X                        O   O   P  E        D        T        R
*T                        L   L   E  R        E        N        M
*----------------------- --- --- -- -------- -------- -------- --------
* FILTER OUT LOGON AND LOGOFF MESSAGES SO OPERATOR NEEDN'T SEE THEM
*----------------------- --- --- -- -------- -------- -------- --------
$OUTPUT/OF                19  36  3
/¬AUTO$LOGON               9  33  3
$LOGOFF                   19  34  3
$DSCONNECT                19  36  3
$RECONNECT                19  36  3
$DIAL                     19  32  3
$DROP                     19  32  3
*----------------------- --- --- -- -------- -------- -------- --------
* SEND REMAINING ASYNCHRONOUS CP MESSAGES TO LOGICAL OPERATOR
*----------------------- --- --- -- -------- -------- -------- --------
                                   3                  DMSPOS   LGLOPR
```

**Figure 54. Example of Entries to Filter Responses to Routine Commands**

In Figure 54, the entries that appear in the "TEXT" field (OUTPUT OF, LOGON, etc.) are the texts contained in the messages that are to be trapped by the programmable operator facility when they are issued by CP.

No userids and nodeids are specified for these entries because they are issued by CP. Because no action routine is specified, the only action taken is the logging of the messages in the current day's log file.

Looking at the last line in Figure 54, you can see that if a type-3 IUCV message is received that does not have a corresponding entry in the routing table, action routine DMSPOS together with the LGLOPR parameter routes the message to the logical operator. In this case, this entry has to be placed after the specific text entries that you want filtered from the message stream. If this entry appeared before the text entries in Figure 54, all type-3 IUCV messages would be routed to the logical operator.

### Controlling Authorization

The routing table determines who is authorized to issue specific commands in the programmable operator facility. Programmable operator authorization is based entirely on the contents of the routing table. Therefore, controlling authorization is a relatively simple procedure. Authorization checking is done using either the userid, nodeid, the command text, or any combination thereof. This means that a change to any of these entries can result in a change in authorization. This allows an installation to easily tailor the authorization structure to their particular needs because only the entries in the routing table need to be changed, and not the action routines.

When a userid and nodeid *are not* specified for a routing table entry, all users are authorized to match that entry and to use the function that it describes. Figure 55

shows an example of unrestricted authorization for the FEEDBACK command. A message sent with the FEEDBACK command is passed to module DMSPOR, which supports most of the programmable operator commands. The TOFB parameter invokes the proper action routine contained in module DMSPOR that writes the message to the FEEDBACK file. (See "The Feedback File" later in this section or the *VM/SP Operator's Guide* for more information on the FEEDBACK command.)

**Note:** In the following examples, the TYPE (IUCV message class) field is left blank to allow the FEEDBACK (or FB) command to be issued with any class of IUCV message. The ECOL fields are 9 and 3 because the character string being looked for is FEEDBACK or FB followed by a blank, for example, "FEEDBACK " or "FB " would match.

```
*------------------------ --- --- -- -------- -------- -------- --------
*T                        S   E   T  U        N        A        P
*E                        C   C   Y  S        O        C        A
*X                        O   O   P  E        D        T        R
*T                        L   L   E  R        E        N        M
*------------------------ --- --- -- -------- -------- -------- --------
* PLACE A FEEDBACK MESSAGE IN THE PROP FEEDBACK FILE
*------------------------ --- --- -- -------- -------- -------- --------
/FEEDBACK /                 1   9                       DMSPOR   TOFB
/FB /                       1   3                       DMSPOR   TOFB
   :                        :   :                         :        :
   :                        :   :                         :        :
```

**Figure 55. Example of Uncontrolled Authorization**

Authorization can be restricted to users at a particular network node by specifying only the nodeid. In Figure 56, only users at NODE1 are authorized to issue the FEEDBACK command.

```
*------------------------ --- --- -- -------- -------- -------- --------
*T                        S   E   T  U        N        A        P
*E                        C   C   Y  S        O        C        A
*X                        O   O   P  E        D        T        R
*T                        L   L   E  R        E        N        M
*------------------------ --- --- -- -------- -------- -------- --------
* PLACE A FEEDBACK MESSAGE IN THE PROP FEEDBACK FILE
*------------------------ --- --- -- -------- -------- -------- --------
/FEEDBACK /                 1   9               NODE1    DMSPOR   TOFB
/FB /                       1   3               NODE1    DMSPOR   TOFB
   :                        :   :                 :        :        :
   :                        :   :                 :        :        :
```

**Figure 56. Example of Restricting Authorization by Nodeid**

When a userid and nodeid are specified, only that user at the specified node is authorized to match that entry. In Figure 57, only JOHNDOE at NODE1 and JANEDOE at NODE2 are authorized to place messages in the feedback file.

```
*------------------------ --- --- -- -------- -------- -------- --------
*T                        S   E   T  U        N        A        P
*E                        C   C   Y  S        O        C        A
*X                        O   O   P  E        D        T        R
*T                        L   L   E  R        E        N        M
*------------------------ --- --- -- -------- -------- -------- --------
* PLACE A FEEDBACK MESSAGE IN THE PROP FEEDBACK FILE
*------------------------ --- --- -- -------- -------- -------- --------
/FEEDBACK /                1   9    JOHNDOE   NODE1    DMSPOR   TOFB
/FEEDBACK /                1   9    JANEDOE   NODE2    DMSPOR   TOFB
/FB /                      1   3    JOHNDOE   NODE1    DMSPOR   TOFB
/FB /                      1   3    JANEDOE   NODE2    DMSPOR   TOFB
  :                        :   :       :         :        :        :
  :                        :   :       :         :        :        :
*------------------------ --- --- -- -------- -------- -------- --------
* SEND REMAINING REQUESTS AND COMMANDS TO THE LOGICAL OPERATOR
*------------------------ --- --- -- -------- -------- -------- --------
                                                        DMSPOS   LGLOPR
```

**Figure 57. Example of Restricting Authorization by Userid and Nodeid**

Since the user must explicitly issue the FEEDBACK or FB command to have a message placed in the feedback file, action routine DMSPOR TOFB must be specified in the routing table to carry out the required action. Any user attempting to issue the FEEDBACK command that is not authorized by the routing table in Figure 57 will have their command sent to the logical operator as a message via action routine DMSPOS LGLOPR, as specified by the last record of the routing table.

Additional userids and nodeids may be added to the table to grant authorization to issue these commands. Conversely, userids and nodeids may be removed to revoke authorization.

## Action Routines

Action routines, programs that receive control in response to the match of a message and a routing table entry, handle a particular type of message or command intercepted by the programmable operator facility. A set of action routines is provided with the programmable operator facility. These need no tailoring to provide the installation with the control and function needed to operate the programmable operator facility. Other action routines may be written by the installation as desired to perform specific functions. Thus, the programmable operator facility may be extended simply by the addition of a new action routine.

If an action routine abends, abend error messages are sent to the logical operator and the requester (if any). Control is returned to the point in the programmable operator facility immediately following the action routine call.

**Note:** Programs written in Basic Assembler Language can access the parameter list built by the programmable operator facility. The programmable operator parameters are available in a different fashion for EXEC action routines[9]. For information on the action routine interface, see "The Action Routine Interface" later in this section.

### *Description of Supplied Action Routines*

The action routines supplied with the programmable operator facility are DMSPOR, DMSPOS, and DMSPOL. A parameter must be supplied for module

[9] EXECs may be written using the System Product Interpreter, EXEC 2, or CMS EXEC languages.

DMSPOR. This parameter is the name of the subroutine contained within the module that is invoked to perform an action. DMSPOS may be invoked along with a parameter, which, in this case, is a userid or nickname. DMSPOL may be invoked with a parameter, which is a routing table name.

Note that new action routines are not required to be in this format. The programmable operator facility supports any desired number of action routines. Each one is loaded separately when the programmable operator facility is initialized, or when a LOADTBL command is issued.

The following sections describe the action routines that are supplied with the programmable operator facility. These action routines (or subroutines in the case of DMSPOR) correspond to the programmable operator commands described in the *VM/SP Operator's Guide.*

**DMSPOR - Miscellaneous supplied action routines**

GET - Send the indicated file to an authorized user

This routine sends programmable operator files, such as the log and feedback files, to the user who requested the file via the CMS DISK DUMP command.

QUERY - Return a response to a user query

This routine returns the fileid of the currently active routing table or returns the status of programmable operator node-checking or logging to the user who issued the command.

SET - Change the status of specific functions

This routine stops or resumes the periodic checking of the distributed systems or the host system, or the logging of messages in the log file.

STOP - Stop the programmable operator facility

This routine stops the programmable operator operation after processing currently queued messages. The programmable operator virtual machine returns control to CMS.

TOFB - Write a message to the feedback file

This routine attaches the date and time received to the head of the incoming message and writes it to the feedback file. See "The Feedback File" below for more information.

TOVM - Execute a CP/CMS command

This routine is invoked when the programmable operator CMD command is issued. The text following "CMD" is regarded as the CP or CMS command to be executed in the programmable operator machine, according to the CMS IMPCP and IMPEX settings. The response to the executed CP or CMS command is returned to the authorized user who invoked the CMD command.

Authorized users of the CMD command should be aware of the following:

- Issuing commands that alter or overlay CMS storage, such as CP DEFINE STORAGE, CP IPL CMS, CP SHUTDOWN, and so on, has an adverse effect on the operation of the programmable operator facility.

- Reissuing the PROP command once the programmable operator facility is running causes the programmable operator facility to stop operating correctly. The user must re-IPL CMS and restart the programmable operator facility using the procedure described under "Invoking the Programmable Operator Facility".

- Issuing commands that cause a VM READ or CP READ (interactive commands such as the DDR command) stop the operation of the programmable operator facility. The programmable operator facility must then be restarted in the manner described under "Invoking the Programmable Operator Facility".

- Line editing characters (pound sign (#), for example), as defined by the CP TERMINAL command, are not recognized as line editing characters by the programmable operator facility.

- The CMS immediate commands (e.g. HB, HI, HO, HT, HX, RO, RT, SO, TE, and TS) are not recognized by the programmable operator facility. If a user issues any of these commands, he receives an "UNKNOWN CP/CMS COMMAND" response from the programmable operator facility.

- System and user synonyms for EXECs are not recognized by the programmable operator facility.

In general, the programmable operator facility does no checking to ensure or prevent any of the above circumstances from occurring.

**DMSPOS - Route a message**

DMSPOS sends (routes) a message to the user specified in the RTABLE PARAMETER field. The user is identified by a nickname from the CMS userid NAMES file or by a userid. If the user is on another system, identification must be through a nickname. LGLOPR may be specified as a keyword in the PARAMETER field of the RTABLE, which would indicate that DMSPOS should take the value specified in the LGLOPR statement in the RTABLE. This is the default if the parameter field is left blank.

A message longer than 94 characters (including the 19-character programmable operator origin id) is split and sent as multiple messages. The first piece is no more than 94 characters. The remaining pieces are no longer than 91 characters, and preceded by a continuation mark (".. "). This splitting ensures that the message is small enough to be sent through an RSCS network.

If an error occurs because of an invalid target id, for example, the nickname was not in the "userid NAMES" file, the programmable operator attempts to send the message to the logical operator.

Messages are sent with the CMS TELL facility. If the programmable operator virtual machine is authorized (class B), the CP MSGNOH command is used. If the virtual machine is not authorized to use the CP MSGNOH command, then the CP MESSAGE command is used. For more information on the CMS TELL facility, see the *VM/SP CMS Command and Macro Reference.*

**Notes:**

1. Using the CMS TELL facility requires the user to have a SYSTEM NETID file set up.

2. DMSPOS must not be invoked if the logical operator virtual machine is the same as the programmable operator virtual machine. Also, a parameter should not be specified that directs the message to the programmable operator virtual machine.

If the LGLOPR routine tries to send a message to the the logical operator, but for some reason the logical operator's network node is unavailable, LGLOPR detects this condition and stop any further attempts to send that message. The unsent message, although logged in the current day's log file, is not displayed at the logical operator's console.

### DMSPOL - Load a routing table

This routine dynamically loads the routing table indicated by the programmable operator LOADTBL command. The routing table name must be "filename RTABLE", where "filename" can be any name that conforms to CMS file naming conventions. Although the routing table name specified with the LOADTBL command takes precedence, it is also possible to specify in a routing table the filename of the table to be loaded as a parameter to the action routine. (This can be used as a default.) Therefore, any message selected by the system programmer can cause a new RTABLE to be loaded. Also, the programmer can change the LOADTBL default of "PROP" to whatever is desired without changing the LOADTBL action routine.

**Note:** With the loading of the routing table done by a separate action routine, it is possible for the other routines, DMSPOR and DMSPOS and any user-written routines, to be replaced when a LOADTBL occurs. This permits changes to action routines other than DMSPOL to be made dynamically without stopping the programmable operator.

## The Log File

Every incoming message that the programmable operator facility receives from CP or other virtual machines is put into a CMS disk file referred to as the log file if LOGGING is not OFF. All error messages and command responses generated by the programmable operator facility are also put in the log file if LOGGING is set to ALL. Each message is identified by the date and time received. The userid and nodeid appear only if the text was sent via a CP MSG, SMSG, WNG or sent using SCIF (Single Console Image Facility). Log entries generated and logged by the programmable operator have a userid of PROP. The log file has the following format:

```
col 1    col 10   col 19   col 28     col 39
|        |        |        |          |
V        V        V        V          V
yy/mm/dd hh:mm:ss[userid   nodeid]:   text
```

The log file contains variable length records. The maximum record length that the programmable operator facility can place in the log file is 132 characters. Because the prefix uses 38 of the 132 characters, the text can be only 94 characters long. Therefore if the text of a message exceeds the maximum length of 94 characters the overflow is continued on the next record. This continued record has the same prefix as the preceding record, with no colon preceding the text.

A separate log file is started for each day. The name of the file is:

```
LGyymmdd nodeid A5
```

*where*:

yy is the current year

mm is the current month

dd is the current day

nodeid is the current RSCS nodeid of the system on which the programmable operator facility is running.

When the programmable operator facility is started, stopped or the debug mode is changed, a record is written to the log file. The messages written to the file have the normal log prefix and a text corresponding to the changed function. Generally, responses to the programmable operator *console* commands are written to the log file when LOGGING is set to ON or ALL. It is also possible to have responses from the programmable operator commands written to the log file. See the LOGGING statement of the routing table or the programmable operator SET command for more information. Note that when LOGGING is set to ALL, the log file may be used as an alternative to spooling the virtual console. When node-checking is in effect, by having PROPCHK or HOSTCHK statements in the RTABLE, if a node changes status from UP to DOWN or vice versa, a message is also written to the log file.

If a virtual machine resource limit is reached, such as "disk-full", it may not be possible to write another record to the programmable operator facility log file. If this happens, a user-written EXEC is invoked to perform whatever recovery action the user thinks is desirable or necessary. The user EXEC must have the filename of PROPLGER. See "LOG Error Exit" later in this section.

Any user authorized in the active routing table can obtain the log file as a reader spool file by using the programmable operator GET LOG command. Messages can be placed in the log file by authorized users by using the programmable operator LOG command with no other action being taken. (See the *VM/SP Operator's Guide* for information on the programmable operator facility commands.)

An old log file can be purged by any user authorized in the active routing table to use the programmable operator CMD command by issuing the CMS ERASE command.

### *Ensuring a Complete Log*

When the programmable operator facility routes a message to the logical operator, the message contains the userid of the sender. The operator, in responding to the message, may choose to send a message directly to the user without going through the programmable operator facility. However, if this is done, the message is not logged in the log file. To ensure that these messages are logged, the operator should send the message to the user through the programmable operator facility by using the programmable operator facility CMD command. See the "Programmable Operator Facility Commands" section of the *VM/SP Operator's Guide* for information on the programmable operator facility commands.

Whether the message was sent through the programmable operator or not has little significance to the user. However, so that the messages received by the user always have the same id (the programmable operator facility id), the message should always be sent from the logical operator through the programmable operator facility.

## The Feedback File

The feedback file is another CMS disk file (named FEEDBACK nodeid A5) that the programmable operator facility manages. The feedback file, unlike the log file, is not automatically written by the programmable operator facility. Authorized users can write time stamped notes and complaints about the operation of the system to this feedback file. To write a notice to the feedback file, you, as a user, must explicitly use the FEEDBACK (or FB) command. An example of such a message is

```
M OP FEEDBACK RESPONSE TIME WAS SLOW DURING MORNING SHIFT.
```

Because the feedback file is normally smaller than the log file, it is easier for the personnel in charge of the programmable operator facility's maintenance to review the users' comments and identify when and where particular problems occurred.

Each record in the feedback file is prefixed with the date and time the message was logged along with the sender's userid and nodeid. The feedback file has the following format:

```
col 1    col 10   col 19   col 28     col 39
|        |        |        |          |
V        V        V        V          V
yy/mm/dd hh:mm:ss userid   nodeid:    text
```

The feedback file contains variable length records. The maximum record length that the programmable operator facility can place in the feedback file is 132 characters. Because the prefix uses 38 of the 132 characters, the text can be only 94 characters long. Therefore if the text of a message exceeds the maximum length of 94 characters the overflow is continued on the next record. This continued record has the same prefix as the preceding record, with no colon preceding the text.

Any user authorized in the active routing table can obtain the feedback file as a spool file by using the the programmable operator facility GET FB or GET FEEDBACK command. (See the *VM/SP Operator's Guide* for information on the programmable operator commands.)

An old feedback file can be purged by any user authorized by the active routing table to use the programmable operator CMD command by issuing the CMS ERASE command.

## Installing the Programmable Operator Facility

The VM/SP product contains the file PROPLIB LOADLIB which is the basis for the programmable operator facility. After receiving and installing VM/SP, take the following steps before running the programmable operator facility.

1. Reserve enough mini-disk space to contain the log file(s) and feedback file for the virtual machine that the programmable operator facility will be running in. The amount of space needed depends on the amount of message traffic that will be going through the programmable operator facility, and on the number of comments you expect users to place in the log and feedback files.

The amount of space needed depends on the amount of message traffic that will be going through the programmable operator facility, and on the number of comments you expect users to place in the log and feedback files.

2. The sample routing table is located on the CMS 190 mini-disk. In order to use the sample PROP RTABLE, take the following steps:

```
ACCESS 190 C/A
COPYFILE PROP RTABLE C = = A
```

   This places the sample routing table on a read/write mini-disk accessed by the virtual machine. Edit the sample routing table (PROP RTABLE) to include the functions and authorizations to meet the various needs of the installation as described in the previous section, "The Routing Table". Place the edited file on a mini-disk accessed by the programmable operator facility virtual machine.

3. Optionally, if you have made any changes to the supplied action routines, link the TEXT file to the PROPLIB LOADLIB. The CMSGEND EXEC, using the CMSGEND PROP function, allows user-modified routines to be replaced in the PROPLIB LOADLIB.

   If you have written any additional action routines, use the CMS LKED command to add these routines to the PROPLIB LOADLIB. Copy the PROPLIB LOADLIB from the CMS system disk to a read/write disk because any changes would invalidate the directory entry on the system disk. For example, to link a user-written action routine named ACTIONA to the PROPLIB LOADLIB, you would issue:

```
LKED ACTIONA (LET LIBE PROPLIB
```

   Action routines written in Basic Assembler Language must be put in the PROPLIB LOADLIB. EXEC action routines need not be put in the PROPLIB LOADLIB, but can reside on any minidisk accessible to the programmable operator.

## Routing Table Conversion

The format of the routing table is changed from the initial version of the programmable operator facility documented in VM/SP Release 2. The new format makes the specifications easier and the information clearer. VM/SP Release 2 routing tables are not compatible with the current format and must either be re-generated by hand or converted using PROPRTCV. PROPRTCV is a utility provided to convert old routing tables to the current format. This utility is written using the System Product Interpreter. Using an old RTABLE as input, PROPRTCV creates a new RTABLE, leaving the old one unchanged. When you issue the PROPRTCV command, the conversion proceeds as follows:

1. Generates the appropriate configuration statements at the beginning of the new routing table file. See "Routing Table Entry Formats" earlier in this section.

   - A LGLOPR statement is added using the existing logical operator userid and nodeid. PROPRTCV prompts you to change this information, if you wish.

   - A TEXTSYM statement is added. Select the TEXTSYM characters to be used. The text fields of the file are scanned for these characters. If any of

these characters are found, PROPRTCV informs you and then prompts you for different characters. You can also exit and change the texts that caused the conflict.

- PROPCHK statements are added, if desired. PROPRTCV prompts you for this information.
- A HOSTCHK statement is added, if desired. PROPRTCV prompts you for this information.
- A ROUTE statement is placed after all the above statements have been completed.
- An entry for the new SET command is added with text "/SET /", message type 1, action routine DMSPOR, and parameter SET. PROPRTCV prompts you for any authorization desired for this entry. See "DMSPOR - Miscellaneous supplied action routines" for more information on the SET command.

  **Note:** The SET entry is simply placed after the ROUTE statement. This is probably not where you want it. Move the entry when the routing table conversion is completed.

For each routing table entry, PROPRTCV

2. Encloses the specified text with the blank-separator, (/), for the specified length of the text.
3. Generates a starting column value and an ending column value from the existing displacement and length values.
4. Converts an entry with action routine DMSPOR and parameter TOLGLOPR to the action routine name DMSPOS. PROPRTCV prompts you for the routing target information to be used as the parameter.
5. Converts an entry with action routine DMSPOR and parameter LOADTBL to the action routine name DMSPOL and no parameter.

## Invoking the Programmable Operator Facility

### *Manual Invocation*

Before loading and invoking the programmable operator facility, load CMS in the virtual machine that will be running the programmable operator facility.

Use the PROPST EXEC to manually invoke the programmable operator facility. The PROPST EXEC drops *any* IBM-supplied programmable operator routines that are currently loaded as a nucleus extension, and loads the programmable operator as a nucleus extension. It then invokes the programmable operator facility with the specified RTABLE. If you do not specify a routing table, the RTABLE name defaults to "PROP". You may specify a disconnect parameter to disconnect the programmable operator before it is invoked. The format of the invocation EXEC is as follows:

| PROPST | [ rtable-name ]  [ DISConn ]<br>[ PROP ] |
|---|---|

Optionally, you can take the following steps each time you invoke the programmable operator facility:

1. Issue a FILEDEF command to assign a CMS filename to the PROPLIB LOADLIB file so CMS can read and load from it. This is done with the following command:

   ```
   FILEDEF PROPLIB DISK PROPLIB LOADLIB *
   ```

2. Next, load the programmable operator program as a CMS nucleus extension via the NUCXLOAD command. Issue the command as follows:

   ```
   NUCXLOAD PROP DMSPOP PROPLIB
   ```

   (See the *VM/SP CMS Command and Macro Reference* for more details on the NUCXLOAD command.) These first two steps may be omitted for subsequent invocations as long as you do not:

   - IPL CMS or
   - Have a CMS abend from which the programmable operator does not automatically recover.

Following its loading as a CMS nucleus extension, invoke the programmable operator facility as if it were a CMS command. The format of the invocation is:

```
PROP  [rtable-name]
      [PROP      ]
```

*where*:

rtable-name is the filename of the routing table that is to be used for the programmable operator facility. "PROP" is the default filename of the routing table if no other is specified at invocation.

The action routines named in the default or specified routing table are in turn loaded as CMS nucleus extensions. If the programmable operator facility cannot find an action routine that is named in the routing table, the user receives an error message and is informed of all detectable routing table errors before the programmable operator facility terminates operation. When all of the required action routines have been loaded, a message is typed on the programmable operator's console indicating that the programmable operator facility has started. The programmable operator facility then waits for either an incoming message or a programmable operator console command (STOP and SET are the only valid commands). The operator can disconnect at this point by having specified the DISConn parameter for the PROPST EXEC or by entering CP (pressing the PA1 key or equivalent) and typing CP DISCONN. After the DISCONNECTED message has been written to the console, indicating that the system operator virtual machine is disconnected, the operator can log on to whatever virtual machine he/she is normally supposed to use, for example, the logical operator virtual machine specified in the routing table.

### *Automatic Invocation*

If you wish, you can set up the programmable operator facility to start running when the system is IPLed and to restart automatically in the event of CP system restart. This can be done as follows:

- Place an "IPL CMS PARM AUTOCR" entry for programmable operator's virtual machine in the CP directory. This can be done even if the programmable operator virtual machine is the CP system operator.

- Place the following entry in the PROFILE EXEC of the programmable operator's virtual machine:

```
EXEC PROPST rtable-name [DISConn]
```

  You can precede or follow the invocation of the PROPST EXEC by the invocation of any virtual machine commands that you wish to have executed before or after the programmable operator facility is invoked. Virtual machine commands that are placed after the invocation of the PROPST EXEC are not executed until the programmable operator facility is stopped.

- If you want the programmable operator to run in other than the operator's virtual machine, place an AUTOLOG entry for the programmable operator's virtual machine in the PROFILE EXEC of the system operator or the AUTOLOG user.

- Once this is complete, if the logical operator is not already logged on, he/she should do so on the appropriate system.

The following example shows how to place entries in the CP Directory and the PROFILE EXEC of operator's virtual machine. These entries automatically invoke the programmable operator facility in the operator's virtual machine when the system is IPLed. The userid of the programmable operator virtual machine is "OPERATOR". The default, "PROP RTABLE", is the name of the routing table being used.

CP directory entries:

```
   :
   :
USER OPERATOR password 512K 1024K ABCDEFG
IPL CMS PARM AUTOCR
   :
   :
```

Entries in the PROFILE EXEC of the Operator's virtual machine:

```
   :
   :
EXEC PROPST DISCONN
   :
   :
```

Once these changes (or similar ones) have been made, IPLing the system causes the programmable operator to be automatically invoked in the disconnected system operator virtual machine. After the DISCONNECTED message has been written to the console, indicating that the system operator virtual machine is disconnected, the operator can log on to whatever virtual machine he/she is normally supposed to use, for example, the logical operator virtual machine specified in the routing table.

## Communications Checking

The programmable operator facility can operate either from the host system or from a distributed system in a network or from both sides. Special functions can be performed depending on the ability of the programmable operators to communicate through RSCS Networking. The purpose of these functions is:

- To provide the host operator (logical operator) with timely information and/or action in the event of a break in communication with the programmable operator on one of the network's distributed systems.
- To provide a distributed system with timely information and/or capability for action in the event of a break in communication with the host system.

A programmable operator can periodically check on the link with another system to determine whether it is possible to communicate with that system. The systems to be checked must be identified in the routing table of the programmable operator doing the checking. This may be either the programmable operator at the host system checking on specified distributed systems or a distributed programmable operator checking on communications with its host system. When the programmable operator at the host system is checking on the distributed systems, the programmable operator needs another programmable operator running in the system operator virtual machine on the distributed system. This is not required when a distributed system is checking on the host system. In other words, for "host checking", no programmable operator is required at the host system, but for "distributed system checking" programmable operators must be running at the distributed systems. These various types of checking may be collectively referred to as "node-checking".

Note that the roles of the 'host' and 'distributed' systems need not be strictly defined. For example, a programmable operator may use the PROPCHK function to check communication with *any* other system (node) running a programmable operator in it's system operator virtual machine in the network. With the HOSTCHK function, the system being checked is simply the system defined as the checking system's logical operator, and not necessarily 'the' host system for the network.

The programmable operator facility that has been instructed to check on its distributed system(s) periodically attempts to communicate with those systems by sending a message that causes a response. The programmable operator then waits a specified time for a response. For checking the host system (HOSTCHK), the acknowledgement request goes to the RSCS on the logical operator node. For checking the distributed systems (PROPCHK), it goes to the programmable operator on the distributed system. No response indicates that something has prevented communication between the host and the distributed system(s). Getting a response after being delinquent for a time indicates that communication between the programmable operators has been restored. With the SET command, the user is able to set the checking function ON or OFF.

Any time that the programmable operator detects that a node has exceeded the time allowed for responding, that fact is recorded in the programmable operator log. Also logged is the fact that a node has resumed responding.

When one of these conditions, no response or a late response, is detected, the programmable operator facility invokes one of two EXECs supplied by IBM for this purpose. For checking a distributed system, the PROPPCHK EXEC is invoked. For checking of the host, the PROPHCHK EXEC is invoked.

The programmable operator doing the checking invokes the EXEC. For example, if a programmable operator on a distributed system has a HOSTCHK statement in its routing table, the PROPHCHK EXEC would be invoked if communication with the host system were lost for a long enough period that the request or the response were prevented from getting through. Similarly, if a programmable operator on a host system has a PROPCHK statement in its routing table, that programmable operator would invoke the PROPPCHK EXEC if communication with one of the specified distributed nodes were lost for such a period. These EXECs are supplied as samples only and may be modified or replaced with user-written EXECs, allowing the user to tailor the resulting action(s).

The IBM-supplied PROPHCHK EXEC operates as follows:

- When the logical operator's node fails to respond or resumes responding, type a message on the programmable operator console and send a message to the userid MAINT indicating that communication with the host system has been broken or restored.

The IBM-supplied PROPPCHK EXEC operates as follows:

- For each node that has failed to respond, notify the logical operator that the programmable operator facility is unable to communicate with that particular node (distributed system).
- For each node that has resumed responding from a failed state, notify the logical operator that communication with that node (distributed system) has been reestablished.

## How the Programmable Operator Establishes Communications with IUCV

The programmable operator facility automatically establishes communications with CP through the Inter-User Communications Vehicle (IUCV). When the programmable operator facility is initialized, a CMSIUCV CONNECT, specifying *MSG and an application id of PROP, is issued to establish the communications path with the Message System Service (See the "Message System Service" section earlier in this manual). This allows the programmable operator program to read and evaluate messages directly from CP.

Several CP command settings determine the types of messages that the programmable operator facility can receive. The programmable operator facility issues these SET commands when initializing, and resets them when terminating. The commands issued during initialization are:

```
SET MSG IUCV
SET WNG IUCV
SET SMSG IUCV
SET EMSG IUCV
SET IMSG IUCV
SET CPCONIO IUCV
SET VMCONIO OFF
```

VMCONIO is set OFF so that any messages produced by CMS or the programmable operator during initialization of the programmable operator facility are typed on its virtual machine console. If VMCONIO was set to IUCV, such data would be trapped by IUCV and not displayed.

When the programmable operator STOP command is issued, the following SET commands are issued:

```
SET MSG ON
SET WNG ON
SET SMSG OFF
SET EMSG ON
SET IMSG ON
SET CPCONIO OFF
SET VMCONIO OFF
```

Then, after the existing messages are handled, the IUCV connection is severed using the IUCV SEVER function.

Some other virtual machine settings that the programmable operator facility modifies are "SET RUN ON", "SET TIMER REAL", and "TERMINAL MODE VM" at initialization, and "SET RUN OFF" and "SET TIMER ON" at termination. "SET RUN ON" is issued to ensure that the programmable operator is not held up in CP console function mode for excessive periods of time, either because of some operator command entry or because of logging on to a disconnected programmable operator virtual machine. "TERMINAL MODE VM" is to ensure that programmable operator console commands are handled correctly.

**Note:** SCIF (Single Console Image Facility) operation supersedes IUCV Message System Service operation. If the programmable operator virtual machine has a SCIF secondary user, messages would be sent via SCIF to the secondary user rather than handled by the programmable operator virtual machine through the IUCV Message System Service. However, the programmable operator facility may be a SCIF secondary user for another virtual machine. For example, this can be used to control the operation of a guest operating system running in another virtual machine. In this case, SCIF messages is presented to the programmable operator virtual machine as IUCV message-type 8.

## Message Output Format

The messages and responses from the programmable operator facility are sent via the CP MSGNOH command if the programmable operator virtual machine has user class B authorization. Otherwise, the CP MESSAGE command is used. Regardless of which message command is used the messages from another user that are routed to the logical operator are prefixed with the userid and nodeid of the originating user.

The format of these messages appears as follows:

```
col 1     col 10     col 20
|         |          |
V         V          V
userid    nodeid:    text
```

Messages that the programmable operator facility sends as responses to the issuer of a programmable operator command or an asynchronous message to the CP operator originating at the programmable operator virtual machine have no such prefix.

## Exit EXECs

### *Exit EXEC Interface*

The programmable operator facility exit EXECs have the same parameter list provided as an EXEC action routine, with the exception that no RTABLE parameter field value and no message text are stacked for the EXEC. When an exit EXEC is called, contents of the program stack depend upon which exit is being invoked. Descriptions of the stack contents for the different types of exits follow:

**Notes:**

1. Some of the parameter values have no meaning for a particular exit EXEC and their use is left to the discretion of the EXEC writer. For example, requester's userid and nodeid have no meaning for the communication error EXECs, PROPPCHK and PROPHCHK.

2. The programmable operator facility does not trap VMCONIO-type or CP EMSGs produced by exit EXECs as it does for action routines.

### *Communication Error Exit*

The PROPPCHK EXEC is invoked when the programmable operator facility determines that communication with a node that is being checked has changed status. When this occurs, the following information is stacked, LIFO, for the EXEC.

1. Entries having the format

   ```
   "nodeid UP"   or   "nodeid DOWN"
   ```

   *where*:

   nodeid — is the RSCS nodeid of a node that has changed communication status.

   UP — indicates that the node had not been responding and has resumed responding to acknowledgement requests.

   DOWN — indicates that the node had been responding and has ceased responding.

2. Total number of nodeid entries stacked.

The PROPHCHK EXEC is invoked when the programmable operator determines that communication with the logical operator node (if it is being checked) has changed status. If the status has changed, a line is stacked LIFO for the EXEC. The line is either "nodeid UP" or "nodeid DOWN", where "nodeid" is the RSCS nodeid of the logical operator and "UP" and "DOWN" have the same meaning as for PROPPCHK.

### *LOG Error Exit*

If a virtual machine resource limit is reached, such as "disk-full", it may not be possible to write another record to the programmable operator facility log file. If this happens, a user-written EXEC is invoked to perform whatever recovery action the user thinks is desirable or necessary. The user EXEC must have the filename of

PROPLGER. The programmable operator facility stacks (LIFO) the error code received from the CMS FSWRITE function. The programmable operator performs the following actions depending on the return code from the EXEC.

RC = 0 recovered from error. The programmable operator facility should retry logging. If it is still unable to log, an error message is sent.

RC = 4 unable to do recovery. The programmable operator facility should send an error message.

The error message is sent to the logical operator. If the PROPLGER EXEC cannot be found, the programmable operator facility acts as if RC = 4 has been returned thus, an error message is sent to the logical operator. Whatever action is taken, the programmable operator facility continues operation. The IBM-supplied sample PROPLGER EXEC:

- Closes the current log file
- Sends the last two log files to the logical operator
- Erases the last two log files.

If the same logging error occurs on two successive logging attempts, (for example, two consecutive incoming messages cause the same logging error) the programmable operator sets LOGGING to "OFF". This prevents unpredictable looping in some situations. Note, though, that the logical operator may receive only two error messages when logging errors occur.

## Problem Determination - Debug Mode

Debug mode is used to perform problem determination on the programmable operator program itself. It allows responses to commands issued from the programmable operator virtual machine console to be returned back to the console without being intercepted by the programmable operator program. This permits any CP command (for example, CP TRACE and ADSTOP commands), to be issued without having its response trapped by the programmable operator program.

SET DEBUG ON may be used after the programmable operator facility responds with the message:

```
PROP RUNNING - ENTER ' STOP ' TO TERMINATE
```

indicating that the programmable operator facility is running and operational. The programmable operator facility then responds with the message:

```
PROP IS RUNNING IN DEBUG MODE
```

which is also written to the log file. Once in debug mode, the programmable operator facility waits to receive messages from another virtual machine, or for the system programmer to enter input from the console. Since only two commands are accepted from the programmable operator virtual machine console (STOP and SET), to issue any CP commands the System Programmer must enter the CP environment (using the PA1 key). Otherwise, the commands are intercepted and rejected as invalid programmable operator commands.

Conversely, pressing the PA1 key or issuing the "BEGIN" command returns control to the programmable operator facility. From this environment, issuing SET DEBUG OFF causes the programmable operator facility to return to its normal function of trapping messages.

## The Action Routine Interface

### *Action Routine Call Interface*

Action routines are loaded by the programmable operator facility as CMS nucleus extensions. As a result, they must be invoked by the programmable operator facility as CMS commands via SVC 202. Also, addresses cannot be resolved between separate nucleus extensions; they must be passed dynamically if they are desired.

### *Action Routine Parameter Interface*

An installation can write additional action routines in Basic Assembler Language. Action routines may also be written as EXECs. Programs written in Basic Assembler Language can access the parameter list built by the programmable operator facility. (The programmable operator parameters are available in a different fashion for EXEC action routines--see below.) The parameter list contains a list of addresses pointing to data that may be significant to the action routine invoked. The programmable operator facility then passes the address of the list as a parameter when it invokes the required action routine. See *VM/SP Data Areas and Control Block Logic, Volume 2* for descriptions of the DSECTs mentioned below.

The register conventions used for invoking an action routine are:

- Register 1 points to a list of eight-byte tokens (CMS PLIST) containing the following information:

  TOKEN 1 Contains the command name (action routine name).

  TOKEN 2 Contains two fullwords. These fullwords contain the following:

  Fullword 1 - Contains the address of the PROP common area as described by the PROPCOM DSECT.

  Fullword 2 - Contains the address of a list of addresses that point to data that may be needed by the action routine. This list is described by the PARMLIST DSECT.

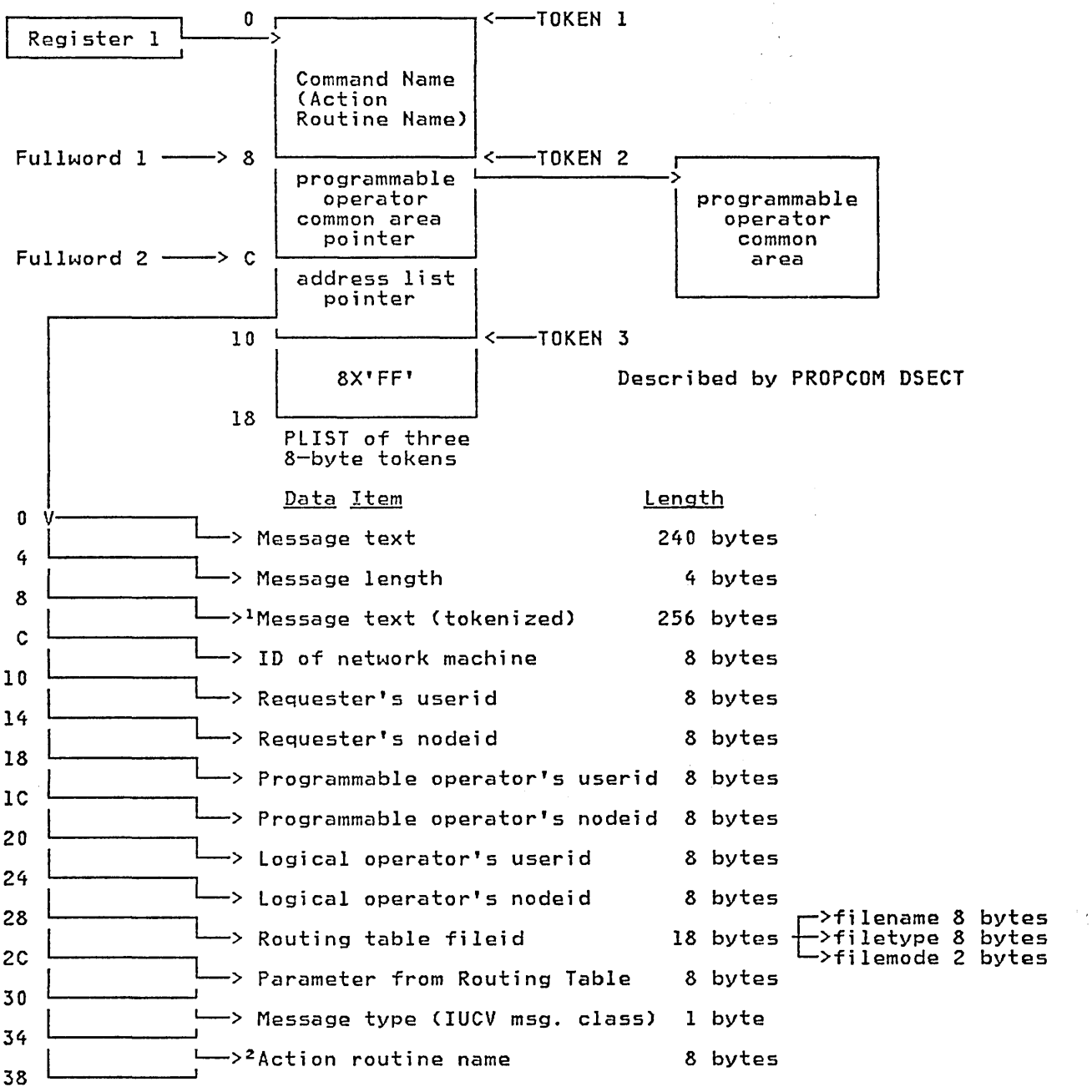
  TOKEN 3 Contains eight X'FF's to mark the end of the parameter list.

- Register 13 points to a standard OS eighteen word save area.

- Register 14 points to the address that receives control when the action routine completes processing, i.e. the address to which the action routine must return control.

- Register 15 points to the action routine entry point and may be used as a base register.

Figure 58 offers a graphic representation of the previous discussion. It illustrates the data areas that can be accessed through Register 1.




Address List described by PARMLIST DSECT

[1]In addition to the original message text, the message text is also provided in CMS tokenized form (eight-byte tokens followed by 8X'FF').

[2]The high order bit (X'80') of the last fullword of this list of addresses is set to one to indicate that it is the last entry in the list according to standard OS linkage conventions.

**Figure 58. Register Conventions for Invoking an Action Routine**

## *EXEC Action Routines*

Action routines may also be written as EXECs[10]. The programmable operator parameters are available in a different fashion for EXEC action routines. The method is described below.

1. Having determined that the action routine is an EXEC, the programmable operator facility calls the action routine accordingly.

2. The following information is passed as parameters (arguments) on the EXEC invocation, in this order:

   - Requester's userid
   - Requester's nodeid
   - Logical operator's userid
   - Logical operator's nodeid
   - Message type code
   - The programmable operator facility's userid
   - The programmable operator facility's nodeid
   - Networking machine userid
   - RTABLE filename

3. The following parameters are stacked LIFO for the EXEC in this order:

   - RTABLE PARAMETER field contents
   - Message text

In order to facilitate handling by an EXEC, if the requester is CP, the requester's userid and nodeid are "CP".

## *Writing Action Routines*

How an action routine is written using Basic Assembler Language depends on the function(s) that the action routine performs and the conditions under which it runs. Since the programmable operator can use message content and message origin to determine which action routine to call, it may not be necessary for the action routine to check any further conditions. However, by using the PARMLIST supplied by the programmable operator facility, the action routine may obtain additional information about the message. Each entry in the PARMLIST points to some item of data about the message just received or about the programmable operator environment.

As described in the preceding section, "Action Routine Parameter Interface," the information initially provided to the action routine is in the form of a CMS

10 EXECs may be written using the System Product Interpreter, EXEC 2, or CMS EXEC languages.

tokenized PLIST. By loading the second fullword of the second token of that PLIST into a register, the user can establish addressability to the PROP PARMLIST. For example,

```
SAVE   (14,12)              SAVE REGISTERS
LR     R12,R15              LOAD BASE REGISTER
USING  ROUTINEX,R12         ESTABLISH ADDRESSABILITY
L      R2,12(,R1)           LOAD PROP PARMLIST ADDRESS
USING  PARMLIST,R2          PARM ADDRESSABILITY
```

These instructions would be sufficient for many action routines to establish addressability for the action routine and the PROP PARMLIST. The following instructions could then be used to obtain the addresses of the requester's (message originator's) userid and nodeid.

```
L      R4,PARMRUSR          GET REQUESTER'S USERID ADDRESS
L      R6,PARMRNOD          GET REQUESTER'S NODEID ADDRESS
```

The PROP DSECTs, such as PARMLIST, define the above labels. To include the PROP DSECT in the action routine insert the following assembler instruction in the source file for the routine:

```
COPY   PROP
```

In addition, it may be desirable to include the CMS REGEQU macro instruction for register equates. When the action routine is complete, it is necessary to restore registers and branch to the address in register 14, or use the OS RETURN macro.

### *Handling Console I/O in an Action Routine*

The installation must determine how an action routine is to handle console I/O generated by the virtual machine and CP. Normal operation of the programmable operator facility sets VMCONIO to IUCV (by default) before calling an action routine and sends the VMCONIO to the message originator (requester). If it is desired that console I/O (VMCONIO) produced by the action routine be typed on the programmable operator virtual machine console, the action routine must SET VMCONIO OFF. However, because there would not normally be an operator at the programmable operator virtual machine console, an installation can code an action routine to receive and handle VMCONIO instead of allowing the programmable operator to receive it and send it to the requester (message originator). To accomplish this, the action routine can receive the VMCONIO that was generated by using the IUCV RECEIVE function with message type 5 specified as the IUCV target class (TRGCLS). For details on using IUCV, refer to the section on IUCV earlier in this manual.

An example of this may be found in the subroutine CALLARTN of the IBM-supplied module DMSPOP. In order to use IUCV it is necessary to include the CP COPY files IPARML and EQU.

CP generated console I/O (CPCONIO) should be handled differently than above. The CPCONIO setting should not be changed because this could cause the programmable operator facility to miss some asynchronous CP messages.

If the action routine is to receive the responses from CP commands that it issues, it should use the CP Diagnose X'08' support with a command response buffer, rather than attempting to receive it with IUCV. (See "Diagnose Code

X'08' - Virtual Console Function".) The reason for this is that other CP messages can be mixed in with the command response, and therefore the program cannot be assured of receiving its response in consecutive IUCV messages.

If the CP command response is to be typed on the programmable operator's virtual machine console, the action routine should use a CMS function, such as WRTERM, to write the lines in the program's CP command response buffer to the terminal.

# Auxiliary Directories

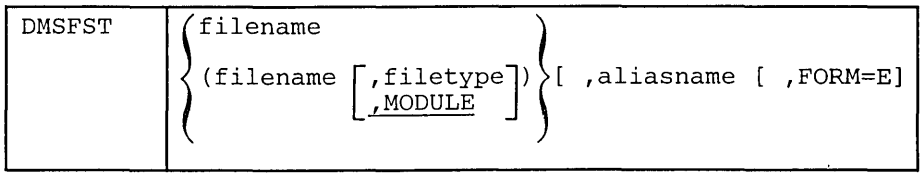
When a disk is accessed, each module that fits the description specified on the ACCESS command is included in the resident directory. An auxiliary directory is an extension of the resident directory and contains the name and location of certain CMS modules that are not included in the resident directory. These modules, if added to the resident directory, would significantly increase its size, thus increasing the search time and storage requirements. An auxiliary directory can reference modules that reside on the system (S) disk; or, if the proper linkage is provided, reference modules that reside on any other read-only CMS disk. To take advantage of the saving in search time and storage, modules that are referenced via an auxiliary directory should never be in the resident directory. The disk on which these modules reside should be accessed in a way that excludes these modules.

## How To Add an Auxiliary Directory

To add an auxiliary directory to CMS, the system programmer must generate the directory, initialize it, and establish the proper linkage. Only when all three tasks are completed, can a module described in an auxiliary directory be properly located.

### *Generation of the Auxiliary Directory*

An auxiliary directory TEXT deck is generated by assembling a set of DMSFST macros, one for each module name. The format of the DMSFST macro is:




*where:*

`filename,filetype` is the name of the module whose File Status Table (FST) information is to be copied.

`aliasname` is another name by which the module is to be known.

`FORM=E` specifies that 64-byte FST entries are to be generated rather than 40-byte entries. Either length FST entry will operate correctly on basic CMS. However, the 40-byte form will not contain such information as date/time after initialization by GENDIRT.

### *Initializing the Auxiliary Directory*

After the auxiliary directory is generated via the DMSFST macro, it must be initialized. The CMS GENDIRT command initializes the auxiliary directory with the name and location of the modules to reside in an auxiliary directory. By using the GENDIRT command, the file entries for a given module are loaded only when the module is invoked. The format of the GENDIRT command is:

| GENDIRT | directoryname [targetmode [sourcemode]] |
|---|---|

*where*

`directoryname`
is the entry point of the auxiliary directory.

`targetmode`
is the mode of the disk containing the modules referenced in the auxiliary directory. The letter is the mode of the disk containing the modules at execution time, not the mode of the disk at the initialization of the directory. At directory creation, all modules named in the directory being generated must be on either the A-disk on a read-only extension or on the disk specified in the sourcemode parameter. The default value for targetmode is S, the system disk. It is your responsibility to determine the usefulness of this operand at your installation and to inform users of programs using auxiliary directories of the proper method(s) of access.

`sourcemode`
is the mode of the disk that contains the modules or files when the GENDIRT command is issued. If not specified, 'A' is the default.

## *Establishing the Proper Linkage*

The CMS module, DMSLAD, entry point DMSLADAD, must be called by a user program or interface to initialize the directory search order. The subroutine, DMSLADAD, must be called via an SVC 202 with register 1 pointing to the appropriate PLIST. The disk containing the modules listed in the auxiliary directory must be accessed as the mode specified, or implied, by the GENDIRT command before the call is issued. If the GENDIRT command has not been used, the user receives the message: "File not found" or "Error reading file."

The coding necessary for the call is:

```
LA    R1,PLIST
SVC   202
DC    AL4(error return)
```

This call must be executed before the call to any module that is to be located via an auxiliary directory.

The PLIST should be:

```
PLIST  DS    0F
       DC    CL8'DMSLADAD'
       DC    V(directoryname)
       DC    F'0'
```

The auxiliary directory is copied into nucleus free storage. The Active Disk Table (ADT) for the targetmode expressed or implied by the GENDIRT command is found and its file directory address chain (ADTFDA) is modified to include the nucleus copy of the auxiliary directory. A flag, ADTPSTM, in ADTFLG2 is set to indicate that the directory chain has been modified.

The address of the nucleus copy of the auxiliary directory is saved in the third word of the input parameter list and the high order byte of the third word is set to X'80' to indicate that the directory search chain was modified and that the next call to DMSLADAD is a clear request.

To reset the directory search chain, a second call is made to DMSLADAD using the modified PLIST. DMSLADAD removes the nucleus copy of the auxiliary directory from the chain and frees it. DMSLADAD does not, however, restore the caller's PLIST to it initial state.

### Error Handling and Return Codes

An error handling routine should be coded to handle nonzero return codes in register 15. When register 15 contains 1 and the condition code is set to 2, the disk specified by the targetmode operand of the GENDIRT command was not accessed as that mode.

When register 15 contains 2 and the condition code is set to 2, the disk specified by the targetmode operand of the GENDIRT command has not previously had its file directory chains modified; therefore, a call to DMSLADAD to restore the chain is invalid.

## An Example of Creating an Auxiliary Directory

Consider an application called PAYROLL consisting of several modules. It is possible to put these modules in an auxiliary directory rather than in the resident directory. It is further possible to put the auxiliary directory on a disk other than the system disk. In this example, the auxiliary directory is placed on the Y-disk.

First, generate the auxiliary directory TEXT deck for the payroll application using the DMSFST macro:

```
PAYDIRT  START   0
         DC      F'40'  LENGTH OF FST ENTRY
         DC      A(DIRTEND-DIRTBEG) SIZE OF DIRECTORY
DIRTBEG  EQU     *
         DMSFST  PAYROLL1
         DMSFST  PAYROLL2
         DMSFST  PAYROLL3
         DMSFST  PAYFICA
         DMSFST  PAYFEDTX
         DMSFST  PAYSTATE
         DMSFST  PAYCITY
         DMSFST  PAYCREDU
         DMSFST  PAYOVERT
         DMSFST  PAYSICK
         DMSFST  PAYSHIFT
         DC      2A(0) POINTER TO NEXT FST BLOCK
DIRTEND  EQU     *
         END
```

[1]Note: F'64' should be used if FORM=E is specified on DMSFST macro.

In this example, the payroll control program (PAYROLL), the payroll auxiliary directory (PAYDIRT), and all the payroll modules reside on the 194 disk.

In the payroll control module (PAYROLL), the subroutine DMSLADAD must be called to establish the linkage to the auxiliary directory. This call must be executed before any call is made to a payroll module that is in the PAYDIRT auxiliary directory.

```
          LA    R1, PLIST
          SVC   202
          DC    AL4(ERRTN)

PLIST     DS    0F
          DC    CL8'DMSLADAD'
          DC    V(PAYDIRT)
          DC    F'0'
```

Next, all payroll modules must have their absolute core-image files generated and the payroll auxiliary directory must be initialized. In the example, the payroll control module (PAYROLL) is given a mode number of 2 while the other payroll modules are given a mode number of 1. When the PAYROLL program is finally executed, only the files on the 194 disk with a mode number of 2 are accessed. This means only the PAYROLL control program (which includes the payroll auxiliary directory) will be referenced from the resident directory. All the other payroll modules, because they have mode numbers of 1, are referenced via the payroll auxiliary directory.

The following sequence of commands create the absolute core-image files for the payroll modules and initialize the payroll auxiliary directory.

```
ACCESS 194 A
LOAD PAYROLL PAYDIRT
GENMOD PAYROLL          (now the auxiliary directory is included
                        in the payroll control module, but it is
                        not yet initialized.)
LOADMOD PAYROLL
INCLUDE PAYROLL1
GENMOD PAYROLL1         (this sequence of three commands is
   .                    repeated for each payroll module called
   .                    by PAYROLL.)
   .
LOADMOD PAYROLL
INCLUDE PAYSHIFT
GENMOD PAYSHIFT

LOADMOD PAYROLL
GENDIRT PAYDIRT Y
GENMOD PAYROLL MODULE A2
```

When it is time to execute the PAYROLL program, the 194 disk must be accessed as the Y-disk (the same mode letter as specified on the GENDIRT command). Also, the 194 disk is accessed in a way that includes the PAYROLL control program in the resident directory but not the other payroll modules. This is done by specifying a mode number of 2 on the ACCESS command.

```
ACCESS 194 Y/S * * Y2
```

Now, a request for a payroll module, such as PAYOVERT, can be successfully fulfilled. The auxiliary directory will be searched and PAYOVERT will be found on the Y-disk.

**Note:** A disk referred to by an auxiliary directory must be accessed as a read-only disk.

## Assembler Virtual Storage Requirements

The minimum size virtual machine required by the assembler is 256K bytes. However, better performance is generally achieved if the assembler is run in 320K bytes of virtual storage. This size is recommended for medium and large assemblies.

If more virtual storage is allocated to the assembler, the size of buffers and work space can be increased. The amount of storage allocated to buffers and work space determines assembler speed and capacity. Generally, as more storage is allocated to work space, larger and more complex macro definitions can be handled.

You can control the buffer sizes for the assembler utility data sets (SYSUT1, SYSUT2, and SYSUT3), and the size of the work space used during macro processing, by specifying the BUFSIZE assembler option. Of the storage given, the assembler first allocates storage for the ASSEMBLE and CMSLIB buffers according to the specifications in the DD statements supplied by the FILEDEF for the data sets. It then allocates storage for the modules of the assembler. The remainder of the virtual machine is allocated to utility data set buffers and macro generation dictionaries according to the BUFSIZE option specified:

BUFSIZE(STD):
: 37 percent is allocated to buffers, and 63 percent to work space. This is the default if you do not specify any BUFSIZE option.

BUFSIZE(MIN):
: Each utility data set is allocated a single 790-byte buffer. The remaining storage is allocated to work space. This allows relatively complex macro definitions to be processed in a given virtual machine size, but the speed of the assembly is substantially reduced.

## Overlay Structures

An overlay structure can be created in CMS in two different ways, although CMS has no overlay supervision. For descriptions of all the CMS commands mentioned, see the *VM/SP CMS Command and Macro Reference.*

### *Prestructured Overlay*

A prestructured overlay program is created using the LOAD, INCLUDE, and GENMOD commands. Each overlay phase or segment is a nonrelocatable core-image module created by GENMOD. The phases may be brought into storage with the LOADMOD command.

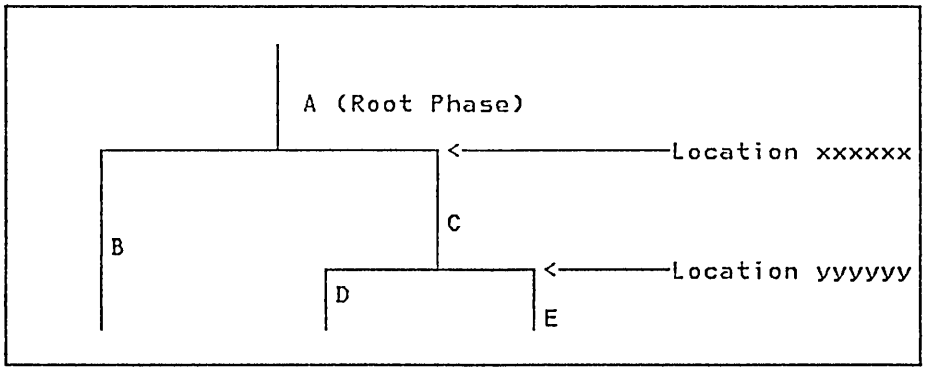



**Figure 59. An Overlay Structure**

The overlay structure shown in Figure 59 could be prestructured using the following sequence of commands (Programs A, B, C, D, and E are the names of TEXT files; the overlay phases will be named Root, Second, Third, etc.):

```
LOAD     A B
GENMOD   ROOT    (FROM A TO B STR)
GENMOD   SECOND  (FROM B)
LOADMOD  ROOT
INCLUDE  C D
GENMOD   THIRD   (FROM C TO D)
GENMOD   FOURTH  (FROM D)
LOADMOD  THIRD
INCLUDE  E
GENMOD   FIFTH   (FROM E)
```

The programmer need not know the storage address where each phase begins. A TEXT file can be made to load at the proper address by reloading earlier phases. In the foregoing example, the command sequences, "LOADMOD ROOT/INCLUDE C D" and "LOADMOD THIRD/INCLUDE E," cause TEXT files C, D, and E to load at the proper addresses.

If the root phase contains address constants to the other phases, one copy of the root must be kept in storage while each of the other phases is brought in by the LOAD or INCLUDE commands without an intervening GENMOD. The root phase is then processed by GENMOD after all address constants have been satisfied. In this case, the programmer must know the address where non-root phases begin (in Figure 59, locations xxxxxx and yyyyyy). The following sequence of commands could be used:

```
LOAD     A B
GENMOD   SECOND  (FROM B)
INCLUDE  C D     (ORIGIN xxxxxx)
GENMOD   THIRD   (FROM C TO D)
GENMOD   FOURTH  (FROM D)
INCLUDE  E       (ORIGIN yyyyyy)
GENMOD   FIFTH   (FROM E)
LOAD     A B
INCLUDE  C D     (ORIGIN xxxxxx)
INCLUDE  E       (ORIGIN yyyyyy)
GENMOD   ROOT    (FROM A TO C STR)
```

The ORIGIN option of the INCLUDE command is used to cause the included file to overlay a previously loaded file. The address at which a phase begins must be a doubleword boundary. For example, if the root phase were X'2BD' bytes long, starting at virtual storage location X'20000', then location xxxxxx would be the next doubleword boundary, or X'202C0'.

The STR option, which is specified in the GENMOD of the root phase, specifies that whenever that module is brought into storage with the LOADMOD command, the Storage Initialization routine should be invoked. This routine initializes user free storage pointers.

At execution time of the prestructured overlay program, each phase is brought into storage with the LOADMOD command. The phases can call LOADMOD. The OS macros LINK, LOAD, and XCTL normally invoke the INCLUDE command, which loads TEXT files. These macros will invoke LOADMOD if a switch, called COMPSWT, in the CMS Nucleus Constant area, NUCON, is turned on.

With COMPSWT set, overlay phases that use LINK, LOAD, and XCTL must be prestructured MODULE files.

### *Dynamic Load Overlay*

The dynamic load method of using an overlay structure is to have all the phases in the form of relocatable object code in TEXT files or members of a TEXT library, filetype TXTLIB. The OS macros, LINK, LOAD, and XCTL may then be used to pass control from one phase to another. The XCTL macro causes the calling program to be overlayed by the called program except when it is issued from the root phase. When issued from the root phase, CMS treats XCTL as it would a LINK macro, adding the new code at the end of the root phase.

The COMPSWT flag in OSSFLAGS must be off when the dynamic load method is used.

# Part 3. Debugging with VM/SP

Part 3, the debugging section, contains the following information:

*Introductory Information*

- How to start debugging
- How to use VM/SP facilities to debug abends, unexpected results, loops, and waits
- Summary of VM/SP debugging tools
- Comparison of CP and CMS debugging tools

*Program Product Information*

- Debugging CP on a virtual machine
- Commands useful in debugging
- Internal trace table
- Restrictions
- Abend dumps
- Reading CP abend dumps
- Control block summary

*Conversational Monitor System Information*

- Debugging commands
- Nucleus load map
- Reading CMS abend dumps
- Control block summary

# Introduction to Debugging

The VM/SP Program Product manages the resources of a single computer such that multiple computing systems appear to exist. Each "virtual computing system&ocq., 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 three separate areas:

1. The Control Program (CP), which controls the resources of the real machine.

2. The virtual machine operating system running under the control of CP, such as CMS, RSCS, OS, or DOS.

3. The problem program, which executes under the control of a virtual machine operating system.

Information explaining how to debug CP or CMS is contained in this book; information explaining how to debug applications programs is in the *VM/SP CMS User's Guide*. For information that explains how to use the VM/370 Interactive Problem Control System (IPCS) for debugging, refer to the *VM/370 Interactive Problem Control System (IPCS) User's Guide.*

If an IPCS problem is caused by a virtual machine operating system (other than CMS and RSCS), refer to the publications pertaining to that operating system for specific information. However, use the CP debugging facilities, such as the CP commands, to perform the recommended debugging procedures discussed in the other publication.

If it becomes necessary to apply a PTF (Program Temporary Fix) to a component of VM/370 or VM/SP, refer to the *VM/SP Installation Guide* for detailed information on applying PTFs.

## How To Start Debugging

Before you can correct any problem, you must recognize that one exists. Next, you must identify the problem, collect information, and determine the cause so that the problem can be fixed. When running VM/SP, you must also decide whether the problem is in CP, the virtual machine, or the problem program.

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. Finally, determine the cause of the problem and correct it.

### *Does a Problem Exist?*

There are four types of problems:

1. Loop
2. Wait state
3. Abend (abnormal end)
4. Incorrect results

The most obvious indication of a problem is the abnormal termination of a program. Whenever a program abnormally terminates, a message is issued. Figure 60 lists the possible abend messages and identifies the type of abend for these messages.

| Message | Type of Abend |
|---|---|
| (Alarm rings)<br>DMKDMP908I SYSTEM FAILURE CODE xxxxxx | CP abend, system dumps to disk. Restart is automatic. |
| DMKCKP900W SYSTEM RECOVERY FAILURE;<br>PROGRAM CHECK<br>DMKCKP901W SYSTEM RECOVERY FAILURE;<br>MACHINE CHECK, RUN SEREP<br>DMKCKP902W SYSTEM RECOVERY FAILURE;<br>FATAL I/O ERROR - NUCL AREA<br>- WARM AREA<br>DMKCKP922W SYSTEM RECOVERY FAILURE;<br>INVALID SPOOLING DATA<br>DMKCKP910W SYSTEM RECOVERY FAILURE;<br>INVALID WARM START CYLINDER<br>DMKCKP911W SYSTEM RECOVERY FAILURE;<br>WARM START AREA FULL | 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;<br>VOLID xxxxxx ALLOCATION ERROR<br>CYLINDER xxx<br>DMKCKT912W SYSTEM RECOVERY FAILURE;<br>VOLID xxxxxx NOT MOUNTED<br>DMKCKV912W SYSTEM RECOVERY FAILURE;<br>VOLID xxxxxx NOT MOUNTED<br>DMKCKS915E PERMANENT I/O ERROR ON<br>CHECKPOINT AREA<br>DMKCKT916E ERROR ALLOCATING SPOOL FILE<br>BUFFERS<br>DMKCKV916E ERROR ALLOCATING SPOOL FILE<br>BUFFERS<br>DMKCKV917E CHECKPOINT AREA INVALID;<br>CLEAR STORAGE AND COLD START | 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. |
| DMKWRM921W SYSTEM RECOVERY FAILURE;<br>UNRECOVERABLE I/O ERROR<br>DMKWRM903W SYSTEM RECOVERY FAILURE;<br>VOLID xxxxxx ALLOCATION ERROR<br>CYLINDER xxx OR PAGE xxxxxx<br>DMKWRM904W SYSTEM RECOVERY FAILURE;<br>INVALID WARM START DATA<br>DMKWRM912W SYSTEM RECOVERY FAILURE;<br>VOLID xxxxxx NOT MOUNTED<br>DMKWRM920W NO WARM START DATA; CKPT<br>START FOR RETRY | 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. |

**Figure 60 (Part 1 of 3). Abend Messages**

| Message | Type of Abend |
|---|---|
| DMKDMP908I SYSTEM FAILURE, CODE xxxxxx<br>DMKCKP960I SYSTEM WARM START DATA SAVED<br>DMKCKP961W SYSTEM SHUTDOWN COMPLETE<br><br>**Optional Messages:**<br><br>DMKDMP905W SYSTEM DUMP FAILURE;<br>PROGRAM CHECK<br>DMKDMP906W SYSTEM DUMP FAILURE;<br>MACHINE CHECK, RUN SEREP<br>DMKDMP907W SYSTEM DUMP FAILURE; FATAL<br>I/O ERROR | CP abend, system dumps to tape or printer. The system stops; the operator must IPL the system to start again.<br><br>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.<br><br>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<br>DAMAGE<br>DMKMCT610W MACHINE CHECK SUPERVISOR<br>DAMAGE<br><br>DMKMCH611W MACHINE CHECK SYSTEM<br>INTEGRITY LOST<br><br>DMKMCT611W MACHINE CHECK SYSTEM<br>INTEGRITY LOST | CP termination with wait state.<br><br>The machine check handler encountered an unrecoverable error with the VM/SP control program.<br><br>The machine check handler encountered an error that cannot be diagnosed; system integrity, at this point, is not reliable. |

**Figure 60 (Part 2 of 3). Abend Messages**

| Message | Type of Abend |
|---|---|
| DMKMCH612W MACHINE CHECK; TIMING FACILITIES DAMAGE; RUN SEREP<br><br>DMKMCT620I MACHINE CHECK; ATTACHED PROCESSOR NOT BEING USED<br><br>DMKMCH622W MACHINE CHECK; MULTIPLE CHANNEL ERRORS<br>DMKACR622W MACHINE CHECK; MULTIPLE CHANNEL ERRORS<br><br>DMKCCH603W CHANNEL ERROR, RUN SEREP, RESTART SYSTEM<br><br>DMKACR603W CHANNEL ERROR, RUN SEREP, RESTART SYSTEM<br><br>DMKCPI955W INSUFFICIENT STORAGE FOR VM/SP<br><br>DMKMCH622W MACHINE CHECK; MULTIPLE CHANNEL ERRORS | An error has occurred in the timing facilities. Probable hardware error.<br><br>A malfunction alert, clock error or instruction processing error occurred on the attached processor. The system continues to run in uniprocessor mode.<br><br>CP termination without automatic restart.<br><br>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.<br><br>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.<br><br>The generated system requires more real storage than is available. CP enters the disabled wait state with code 00D in the PSW.<br><br>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. |
| DMSABN148T SYSTEM ABEND xxx CALLED FROM xxxxxx | CMS abend, system will accept commands from the terminal. Enter the DEBUG command and then the DUMP subcommand to have CMS dump storage on the printer. |
| Others<br>Refer to OS and DOS publications for the abnormal termination messages. | 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. |

**Figure 60 (Part 3 of 3). Abend Messages**

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

Unproductive processing time is another symptom of a problem. This problem is not easily recognized, especially in a timesharing environment.

## *Identifying the Problem*

Two types of problems are easily identified: abnormal termination is indicated by an error message, and unexpected results become apparent once the output is examined. The looping and wait state conditions are not as easily identified.

Unproductive processing time is another symptom of a problem. This problem is not easily recognized, especially in a timesharing environment.

### *Identifying the Problem*

Two types of problems are easily identified: abnormal termination is indicated by an error message, and unexpected results become apparent once the output is examined. The looping and wait state conditions are not as easily identified.

When using VM/SP, you are normally sitting at a terminal You may have a looping condition if your program takes longer to execute than you anticipated. Also, check your output. If the number of output records or print lines is greater than expected, the output may really be the same information repeated many times. Repetitive output usually indicates a program loop.

Another way to identify a loop is to periodically examine the current PSW. 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.

The wait state is also difficult to recognize when at the terminal. If your program is taking longer than expected to execute, the virtual machine may be in a wait state. Display the current PSW on the terminal. Periodically, issue the CP command

```
QUERY TIME
```

and compare the elapsed processing time. When the elapsed processing time does not increase, the wait state probably exists.

Figure 61 helps you to identify problem types and the areas where they may occur.

### *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" describes procedures to follow in determining the cause of various problems that can occur in the Control Program or in the virtual machine. See the *VM/SP CMS User's Guide* for information on using VM/SP facilities to debug a problem program.

If it becomes necessary to apply a Program Temporary Fix (PTF) to a VM/370 or VM/SP component, refer to the *VM/SP Installation Guide* for detailed information on applying PTFs. Figure 62 , Figure 63, and Figure 64 summarize the debugging process from identifying the problem to finding the cause.

| Problem Type | Where Abend Occurs | Distinguishing Characteristics |
|---|---|---|
| Abend | CP abend<br>CMS abend | For a complete discussion of reasons for abends and system programmer's actions, see the CP and CMS 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.<br><br>VM/SP may terminate or reset a virtual machine if a nonrecoverable channel check or machine check occurs in that virtual machine. One of the following messages:<br><br>`DMKMCH616I MACHINE CHECK; USER userid TERMINATED`<br>`DMKCCH604I CHANNEL ERROR; DEV xxx; USER userid; MACHINE RESET`<br><br>is sent to the system operator at the processor console. Also, the virtual user is notified by one of the following messages that his virtual machine was terminated or reset:<br><br>`DMKMCH619I MACHINE CHECK; OPERATION TERMINATED`<br>`DMKCCH606I CHANNEL ERROR; OPERATION TERMINATED` |
| Unexpected Results | CP<br><br><br><br>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 disk or system files (such as spool files) is an error.<br><br>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 VM/SP, a problem exists. |
| Wait | CP<br>LOADER<br>RSCS | For a complete discussion of CP, loader, and RSCS 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 signalling 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 61. VM/SP Problem Types**

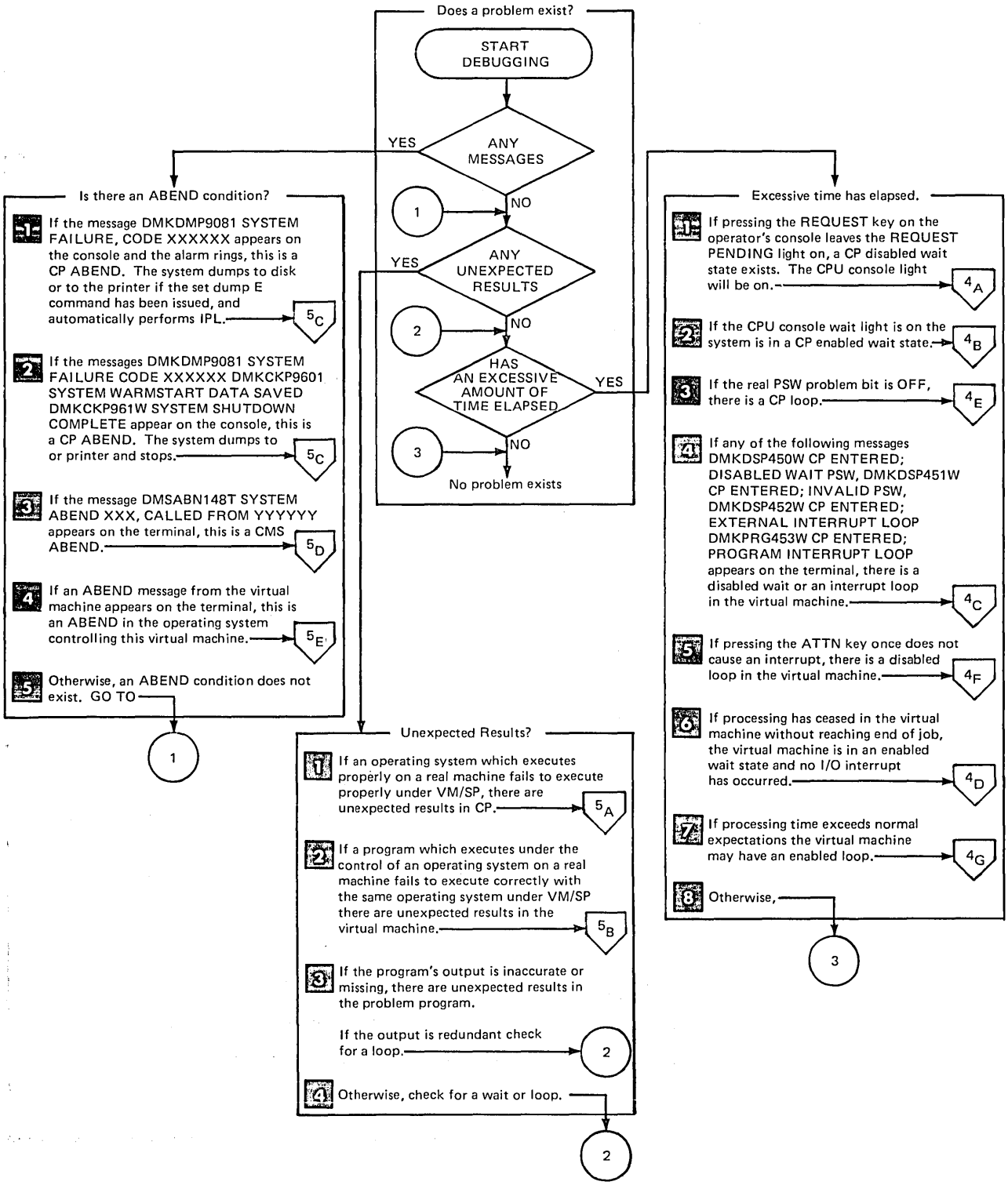



**Figure 62. Does a Problem Exist?**

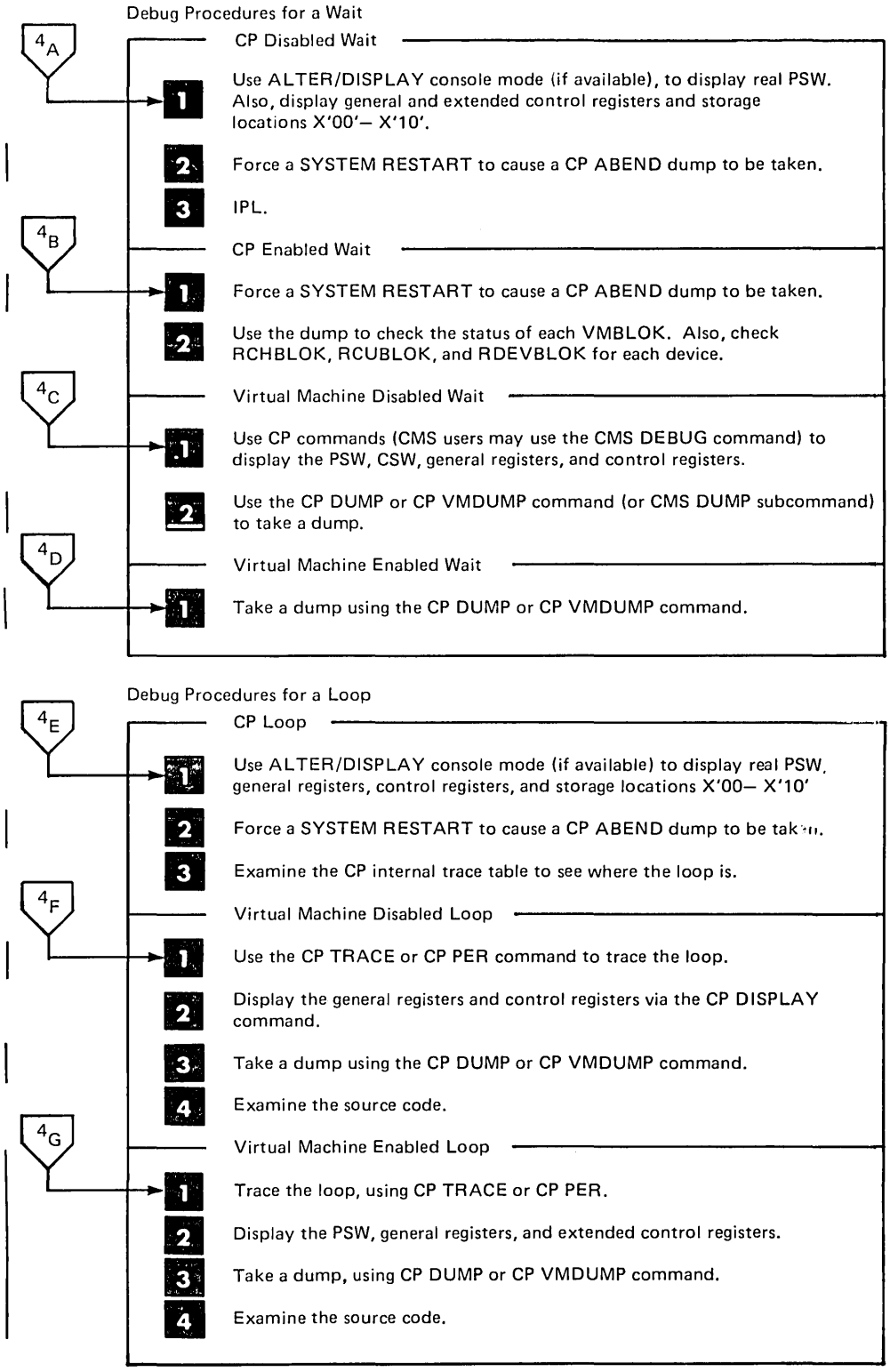



**Figure 63. Debug Procedures for Waits and Loops**

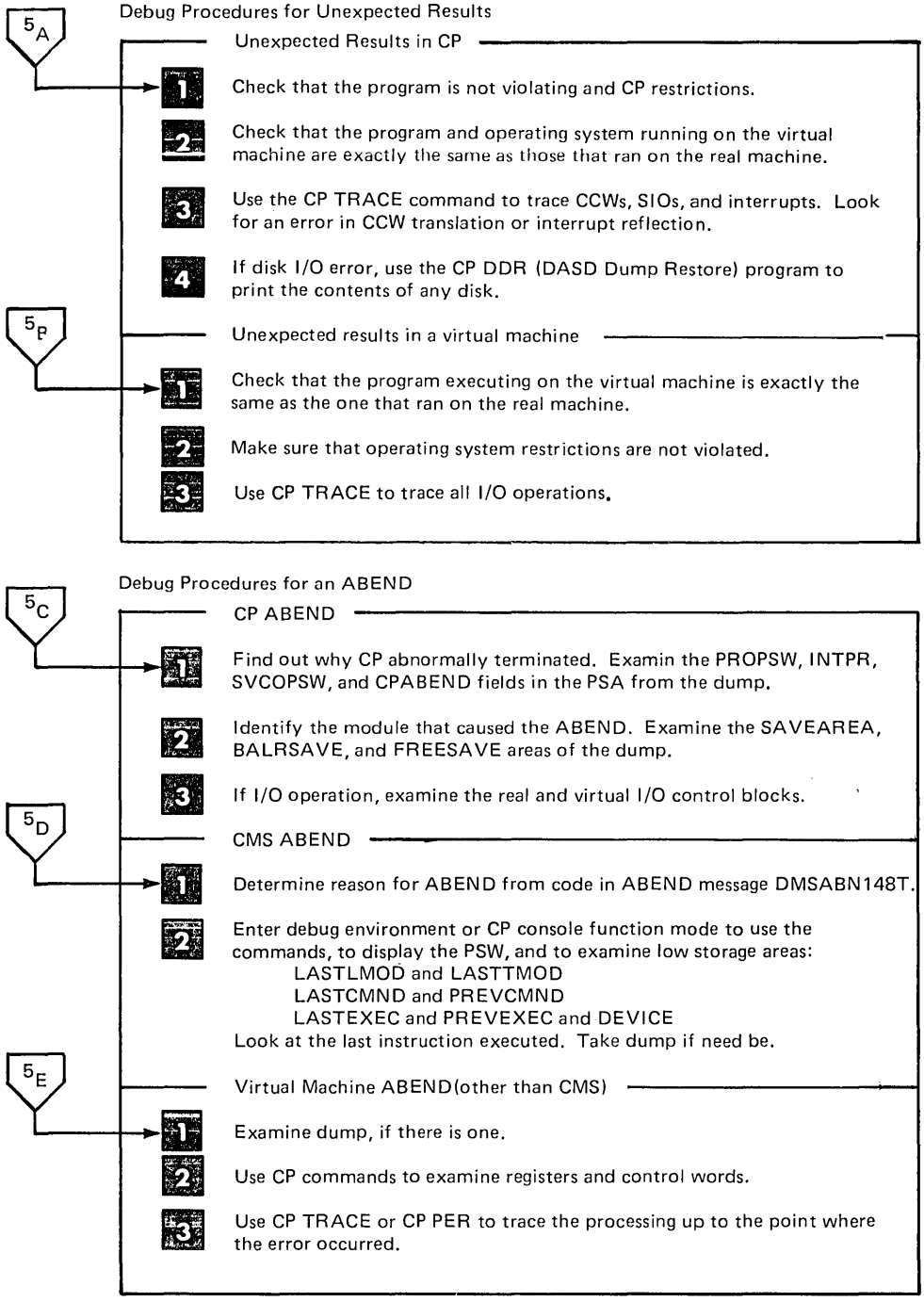



**Figure 64. Debug Procedures for Unexpected Results and an Abend**

## 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 "An Overview of VM/SP Commands That Can Be Used for Debugging" and "CMS Debugging Commands" sections for a detailed description of each debug tool, including how to invoke it.

### *Abend*

Three types of abnormal terminations (ABEND) can occur on VM/SP: CP abends, CMS abends, or virtual machine abends. The following description provides guidelines for debugging each type of ABEND.

#### CP Abend

When the VM/SP Control Program 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 the "Abend Dumps" section for a description of the SET DUMP command.

Use the dump to determine why the control program terminated and then determine how to correct the condition. See the "Reading CP Abend Dumps" discussion for detailed information on reading a CP abend dump.

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

1. Program Check in CP

   Examine the PROPSW and 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 Prefix Storage Area 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 old PSW at location X'08' 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 Data Areas and Control Block Logic, Volume 1*.

Examine the real and virtual control blocks to find the status of I/O operations. Figure 67 on page 504 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. The "CP Internal Trace Table" description tells you how to use the trace table.

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

If the program check 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" 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:

```
DMSAEN148T SYSTEM ABEND xxx CALLED FROM yyyyyy
```

where xxx is the abend code and yyyyyy 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. Because the debug program itself does not alter the sta-

tus of the system, you can use its options 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 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 routine whenever a hardware program exception occurs. If a routine other than a SPIE exit routine is in control, DMSITP issues the message

   ```
   DMSITP141T xxxxxxxx EXCEPTION OCCURRED AT xxxxxx IN
              ROUTINE xxxxxxxx
   ```

   and invokes DMSABN (the abend routine). The abend code is 0Cx, where x is the program exception number (0 through F). The possible programming exceptions are:

   | Code | Meaning |
   |---|---|
   | 0 | Imprecise |
   | 1 | Operation |
   | 2 | Privileged operation |
   | 3 | Execute |
   | 4 | Protection |
   | 5 | Addressing |
   | 6 | Specification |
   | 7 | Decimal data |
   | 8 | Fixed-point overflow |
   | 9 | Fixed-point divide |
   | A | Decimal overflow |
   | B | Decimal divide |
   | C | Exponent overflow |
   | D | Exponent underflow |
   | E | Significance |
   | F | Floating-point divide |

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 DMSABN148T.

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]] |

*Where*:

label
is any valid Assembler language label.

code
is the abnormal termination code (0 through FFF) that appears in the DMSABN148T system termination message.

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

TYPCALL= [SVC / BALR]
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 Abrnorally Terminates***: After an abend, two courses of action are available in CMS. In addition, 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. After using all the DEBUG subcommands that you wish, exit from the debug environment. Then, either issue the RETURN command to return to DMSABN so that abend recovery will occur, or issue the GO command to resume processing at the point the abend occurred.

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 performs the following functions, in sequence:

1. Clears the console input buffer and program stack.

2. Terminates all VMCF activity.

3. Reinitializes the SVC handler, DMSITS, and frees all stacked save areas.

4. Clears the auxiliary directories, if any. Invokes "FINIS * * *", to close all files, and to update the master file directory.

5. Zeroes out EXECFLAG and frees CMS EXEC global storage.

6. Zeroes out the maclib directory pointers.

7. Frees the CMS work area, if the CMS subset was active.

8. Issues the STAE, SPIE, TTIMER, and STAX macros to cancel any outstanding OS exit routines. Frees any TXTLIB, MACLIB, or LINK tables.

9. Calls with a purge plist, all nucleus extensions that have the "SERVICE" attribute defined.

10. Drops all nucleus extensions that do not have the "SYSTEM" attribute. Also drops any nucleus extensions that are in type user storage.

11. Frees all SCBLOCKs associated with SUBCOM.

12. Clears all immediate commands that are not nucleus extensions with the "SYSTEM" attribute. Returns all associated free storage.

13. Frees all storage of type user.

14. Zeroes out all interrupt handler pointers in IOSECT.

15. Turns the SVCTRACE command off.

16. Closes the virtual punch and printer. Closes the virtual reader with the HOLD option.

17. Zeroes out all FCB, DOSCB, and LABSECT pointers.

18. Reinitializes the VSE lock table used by CMS/DOS and CMS/VSAM.

19. Zeroes out all OS loader blocks, and frees the FETCH work area.

20. Disables the CMS IUCV environment, and frees CMS IUCV system storage.

21. Clears all ABNEXIT set and frees storage.

22. 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.

23. Issues a STRINIT if all storage is accounted for.

After abend recovery has been completed, control passes to DMSINT at entry point DMSINTAB to process the new command that was typed in.

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

```
DMSABN149T xxxx DOUBLEWORDS OF SYSTEM STORAGE
           HAVE BEEN DESTROYED
```

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 DEBUG subcommands or CP commands to examine the PSW and specific areas of low storage. For instructions on how to use the CMS debug commands, see "CMS Debugging Commands" in this section. For instructions on how to use the CP commands, see "An Overview of VM/SP Commands that can be Used for Debugging" in this section. See Figure 66 for a comparison of the CP and CMS debugging facilities.

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 PSW subcommand.) Compare the PSW instruction address with the current CMS load map trying 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. 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. EXEC 2 and System Product Interpreter do not update this field.

PREVEXEC — Contains the name of the next-to-last CMS EXEC procedure. 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.

3. Once you have identified the module that caused the abend, examine the specific instruction. Refer to the listing.

4. If you have not identified the problem at this time, take a dump by issuing the debug DUMP subcommand. Refer to "Reading CMS Abend Dumps" for information on reading a CMS dump. If you can reproduce the problem, try the CP or CMS tracing facilities.

**Virtual Machine Abend (Other than CMS)**

The abnormal termination of an operating system (such as OS or DOS) running under VM/SP 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 (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 OS/VS1, OS/VS2, and DOS/VS.

If a dump was taken, it was sent to the virtual printer. Issue a CLOSE command to the virtual printer to have the dump print on the real printer.

The VMDUMP command dumps virtual storage to a specified virtual machine's reader spool file. Installations that have installed the VM/Interactive Problem Control System (IPCS) Extensions program product may use it to process the dump. Other installations may process the dump with a user-written program.

If you choose to run a stand-alone dump program tobdump the storage in your virtual machine, be sure to specify the NOCLEAR option 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
```

appears on the processor console.

If the message:

```
DMKMCT621I  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 VM/SP 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 VM/SP, 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 VM/SP, 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
- The same copy of the operating system (and program)
- The same libraries

If the problem still is not found, look for an I/O problem. Try to reproduce the problem, while tracing all CCWs, SIOs, and interrupts via 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 the Control Program.

### 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 disk files. Non-CMS users may execute any of the utilities included in the operating sys-

tem 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 VM/SP DASD Dump Restore (DDR) program can be invoked by the CMS DDR command in a virtual machine controlled by CMS. The DDR program has the following functions:

*DUMP* -- dumps part, or all, of the data from a DASD device to magnetic tape.

*RESTORE* -- transfers data from tapes created by DDR DUMP to a direct access device. The direct access device to which the data is being restored must be the same type of device as the direct access device originally containing that data.

*COPY* -- copies data from one device to another device of the same type. Data may be reordered by cylinder (or by block number for fixed-block DASDs) when copied from disk to disk. In order to copy one tape to another, the original tape must have been created by the DDR DUMP function.

*PRINT* -- selectively prints the hexadecimal and EBCDIC representation of DASD and tape records on the virtual printer.

*TYPE* -- selectively displays the hexadecimal and EBCDIC representation of DASD and tape records on the terminal.

CMS users should refer to the *VM/SP CMS Command and Macro Reference* for instructions on using the DDR command.

## *Loop*

The real cause of a loop usually is 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 comprise nested loops. Probably, the most difficult case of looping to determine is entry to the loop from 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.

### CP Disabled Loop

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

1. Use the alter/display console mode to display the real PSW, general registers, control registers and storage locations X'00' - X'100'.

   On an attached processor or multiprocessor system, you must add the prefix value for the PSA of the other processor (that is, the processor whose console you are not using) to display, dump, or alter low core storage for the other processor, or use the M or N operand prefixes described under the DCP, DMCP, and STCP commands.

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 examine it.

1. If the cause of the loop is not apparent, examine the CP internal trace table to determine the modules that may be involved in the loop.

2. If the cause is not yet determined, assume that a wild branch caused the loop entry and search the source code for this wild branch.

### 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 and extended control registers via the CP DISPLAY command.

2. Take a dump via the CP DUMP or CP VMDUMP command.

3. Examine the source code.

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 attempting 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 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. CMS users can use the debug DUMP subcommand.

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 the Control Program is in a wait state, its processing ceases.

### CP Disabled Wait

A disabled wait state usually results from a hardware malfunction. During the 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 attempting 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.

Three types of severe machine checks can cause the VM/SP Control Program to terminate or cause a CP disabled wait state.

- An unrecoverable machine check in the control program
- A 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 the control program to terminate.

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

```
DMKCCH603 CHANNEL ERROR, RUN SEREP, RESTART SYSTEM

DMKMCH612W MACHINE CHECK TIMING FACILITIES DAMAGE; RUN SEREP

DMKMCT612W MACHINE CHECK TIMING FACILITIES DAMAGE; RUN SEREP
```

If an unrecoverable machine check occurs in the control program, the message

```
DMKMCH610W MACHINE CHECK SUPERVISOR DAMAGE
```

--or--

```
DMKMCT610W MACHINE CHECK SUPERVISOR DAMAGE
```

appears on the processor console. The control program is terminated and enters a 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. The message

```
DMKMCH611W MACHINE CHECK SYSTEM INTEGRITY LOST
```

--or--

```
DMKMCT611W MACHINE CHECK SYSTEM INTEGRITY LOST
```

appears on the processor console. The control program 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 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

```
DMKCPI955W INSUFFICIENT STORAGE FOR VM/SP
```

appears on the processor console.

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

```
DMKPAG415E  CONTINUOUS PAGING ERRORS FROM DASD xxx
```

appears on the processor console and CP enters the disabled wait state with code 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 the VM/SP system 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 VM/SP 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 registers and the control registers.
2. Force a CP system restart in order to get a system abend dump.
3. IPL the system.

Examine this information and attempt to find what caused the wait. If you cannot find the cause, attempt 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" for specific information on how to analyze a CP dump.

### Virtual Machine Disabled Wait

The VM/SP Control Program 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

DMKDSP451W   CP ENTERED; INVALID PSW

DMKDSP452W   CP ENTERED; EXTERNAL INTERRUPT LOOP

DMKPRG453W   CP ENTERED; PROGRAM INTERRUPT LOOP
```

and enters the console function mode. Use the CP commands to display the following information on the terminal.

- PSW
- CSW
- General registers
- Control registers

Then use the CP DUMP or VMDUMP command to take a dump.

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 commands.

If CMS is running in the virtual machine, the CMS debugging facilities may also be used to display information, take a dump, or trace the processing. The CMS SVCTRACE, CP TRACE, and CP PER commands record different information. Figure 66 compares the CP and CMS facilities for debugging.

### 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.

The Control Program 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 STATE
```

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 Tools

Figure 65 summarizes the VM/SP commands that are useful for debugging programs in a virtual machine. The CP and CMS commands are classified by the function they perform.

| Function | Comments | CP Command | CMS Command |
|---|---|---|---|
| Stop execution at a specified location | Set the address stop before the program reaches the specified address. For CP, ADSTOP allows 1 active address stop; PER allows multiple address stops. CMS allows 16 active address stops. | ADSTOP hexloc<br><br>PER Instruct Range single-addr | DEBUG<br><br>BREAK id {symbol / hexloc} |
| Resume execution | Resume execution where program was interrupted | BEGIN | DEBUG<br>GO |
| | Continue execution at a specific location | BEGIN hexloc | DEBUG<br><br>GO {symbol / hexloc} |
| Dump data | Dump the contents of specific storage locations | DUMP {hexloc1 / Lhexloc1} [{-/:}[hexloc2 / END] / {.}[bytecount / END]]<br>[*dumpid] | DEBUG<br><br>DUMP [symbol1 / hexloc1 / 0] [symbol2 / hexloc2 / * / 176]<br>[ident] |

**Figure 65 (Part 1 of 6). Summary of VM/SP Debugging Tools**

| Function | Comments | CP Command | CMS Command |
|---|---|---|---|
| Dump data | VMDUMP provides the same information that DUMP provides but in a different format; the format is compatible with the VM/Interactive Problem Control System Extensions program product. | VMDUMP {hexloc1 / 0} [ {-/:} [hexloc2 / END] ] [ {.} [bytecount / END] ]<br>[TO * / TO userid / SYSTEM]<br>[FORMAT vmtype]<br>[DSS]<br>[*dumpid] | |
| Display data | Display contents of storage locations (in hexadecimal and EBCDIC) | DISPLAY hexloc1 [ {-/:} [hexloc2 / END] ] [ {.} [bytecount / END] ] | DEBUG X { symbol [ n / length ] / hexloc [ n / 4 ] } |
| | Display contents of storage locations (in hexadecimal and EBCDIC) | DISPLAY Thexlocc1 [ {-/:} [hexloc2 / END] ] [ {.} [bytecount / END] ] | |
| | Display storage key of specific storage locations in hexadecimal | DISPLAY Khexloc1 [ {-/:} [hexloc2 / END] ] [ {.} [bytecount / END] ] | |

**Figure 65 (Part 2 of 6). Summary of VM/SP Debugging Tools**

| Function | Comments | CP Command | CMS Command |
|---|---|---|---|
| | Display general registers | DISPLAY Greg1 [ {-}{:} [reg2 / END] / {.} [regcount / END] ] | DEBUG<br>GPR reg1[reg2] |
| | Display floating point registers | DISPLAY Yreg1 [ {-}{:} [reg2 / END] / {.} [regcount / END] ] | |
| | Display control registers | DISPLAY Xreg1 [ {-}{:} [reg2 / END] / {.} [regcount / END] ] | |
| | Display contents of current virtual PSW in hexadecimal format | DISPLAY PSW | DEBUG<br>PSW |
| | Display contents of CAW | DISPLAY CAW | DEBUG<br>CAW |
| | Display contents of CSW | DISPLAY CSW | DEBUG<br>CSW |
| Store data | Store specified information into consecutive storage locations without alignment | STORE Shexloc hexdata... | DEBUG<br>STORE {symbol / hexloc}<br>hexinfo[hexinfo[hexinfo]] |

**Figure 65 (Part 3 of 6). Summary of VM/SP Debugging Tools**

| Function | Comments | CP Command | CMS Command |
|---|---|---|---|
| | Store specified words of information into consecutive fullword storage locations | STORE {hexloc / Ihexloc} {hexword1[hexword2...]} | |
| | Store specified words of information into consecutive general registers | STORE Greg hexword1 [hexword2...] | DEBUG<br>SET GPR reg hexinfo[hexinfo] |
| | Store specified words of information into consecutive floating-point registers | STORE Yreg hexword1 [hexword2...] | |
| | Store specified words of data into consecutive control registers | STORE Xreg hexword1 [hexword2...] | |
| | Store information into PSW | STORE [PSW hexword1] hexword2 | DEBUG<br>SET PSW hexinfo [hexinfo] |
| | Store information in CSW | | DEBUG<br>SET CSW hexinfo [hexinfo] |
| | Store information in CAW | | DEBUG<br>SET CAW hexinfo |

**Figure 65 (Part 4 of 6). Summary of VM/SP Debugging Tools**

| Function | Comments | CP Command | CMS Command |
|---|---|---|---|
| Trace execution | Trace all instructions, interrupts, and branches | `TRACE ALL` | |
| | Trace SVC interrupts | `TRACE SVC`<br>`PER Instruct DATA 0Axx` | `SVCTRACE ON` |
| | Trace I/O interrupts | `TRACE I/O` | |
| | Trace program interrupts | `TRACE PROGRAM` | |
| | Trace external interrupts | `TRACE EXTERNAL` | |
| | Trace privileged instructions | `TRACE PRIV`<br>`PER Instruct DATA xx`<br>`(PER can trace specific privileged instructions.)` | |
| | Trace all user I/O operations | `TRACE SIO`<br>`PER Instruct DATA xx` | |
| | Trace virtual and real CCWs | `TRACE SIO`<br>`TRACE CCW` | |
| | Trace all user interrupts and successful branches | `TRACE BRANCH` | |
| | Trace successful branches | `PER BRANCH [[INTO]`<br>`into-addr-range]` | |
| | Trace instructions | `TRACE INSTruct`<br>`PER Instruct`<br>`[Range instruction-addr-range]` | |
| | Trace instructions that alter storage | `PER STORE`<br>`[[INTO] storage-addr-range`<br>`[INTO] addr [DATA] hex-data]` | |

**Figure 65 (Part 5 of 6). Summary of VM/SP Debugging Tools**

| Function | Comments | CP Command | CMS Command |
|---|---|---|---|
| | Trace instructions that alter general registers | PER G[reg1] [ {-} [reg2] ; {.} [regcount] ] | |
| | Trace instructions that alter specific bits at specific storage locations | PER MASK<br>[INTO] addr [DATA] mask-field | |
| Trace execution (cont.) | End tracing activity | TRACE END<br>PER END {ALL ; CUrrent ; element-number ; event-type ; traceset name} | SVCTRACE OFF |
| Trace real machine events | Trace events in real machine | MONITOR START CPTRACE | |
| | Stop tracing events in the real machine | MONITOR STOP CPTRACE | |
| | Enable a virtual machine to enter data in CPTRAP file | CPTRAP ALLOWid userid | |
| | Specify selectivity in collecting CPTRAP data | CPTRAP ALL [ON]<br>OFF<br>CPTRAP typenum [Vmblok nnnnn ; DEVaddr nnnn ; COde nnnn ; OFF] | |

**Figure 65 (Part 6 of 6). Summary of VM/SP Debugging Tools**

## Comparison of CP and CMS Facilities for Debugging

If you are debugging problems while running CMS, you can choose the CP or CMS debugging tools. Refer to Figure 66 for a comparison of the CP and CMS debugging tools.

| Function | CP | CMS |
|---|---|---|
| Setting address stops | The CP ADSTOP command can set only one address stop at a time. The PER command can be used to set up multiple address stops. | Can set up to 16 address stops at a time. |
| Dumping storage contents to the printer | The dump is printed in hexadecimal format with EBCDIC translation. The storage address of the first byte of each line is identified at the left. The control blocks are formatted. | The dump is printed in hexadecimal format. The storage address of the first byte of each line is identified at the left. The contents of general and floating-point registers are printed at the beginning of the dump. |
| Displaying the contents of storage and control registers at the terminal | The display is typed in hexadecimal format with EBCDIC translation. The CP command displays storage keys, floating-point registers and control registers. | The display is typed in hexadecimal format. The CMS commands *do not* display storage keys, floating-point registers, or control registers, as the CP command does. |
| Storing information | The amount of information stored by the CP command is limited only by the length of the input line. The information can be fullword aligned when stored. CP stores data in the PSW, but not in the CAW or CSW. However, data can be stored in the CSW or CAW by specifying the hardware address in the STORE command. CP also stores the status of the virtual machine in the extended log out area. | The CMS command stores up to 12 bytes of information. CMS stores data in the general registers but not in the floating-point or control registers. CMS stores data in the PSW, CAW, and CSW. |

**Figure 66 (Part 1 of 2). Comparison of CP and CMS Facilities for Debugging**

| Function | CP | CMS |
|---|---|---|
| Tracing information | CP TRACE traces:<br><br>• All interrupts, instructions, and branches<br>• SVC interrupts<br>• I/O interrupts<br>• Program interrupts<br>• External interrupts<br>• Privileged instructions<br>• All user I/O operations<br>• Virtual and real CCWs<br>• All instructions.<br><br>CP PER provides increased selectivity in tracing the execution of instructions that:<br><br>• Cause successful branches<br>• Alter specific storage locations<br>• Alter specific general registers<br>• Are fetched and executed.<br><br>The CP tracing is interactive. You can stop and display other fields. | CMS traces all SVC interrupts. CMS displays the contents of general and floating-point registers before and after a routine is called. The parameter list is recorded before a routine is called. |

**Figure 66 (Part 2 of 2). Comparison of CP and CMS Facilities**

## An Overview of VM/SP Commands that Can Be Used for Debugging

The VM/SP Control Program provides interactive commands that control the VM/SP 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 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 (as in the case of DOS, MFT, MVT, PCP, CMS, and RSCS) these commands execute as expected. However, when the operating system in the virtual machine itself manipulates virtual storage (as in the case of OS/VS1, OS/VS2, or DOS/VS) these CP commands should not be used.

This section 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 publications:

- VM/SP CP Command Reference for General Users
- VM/SP Operator's Guide
- VM/SP CMS Command and Macro Reference

The following categories of commands are discussed:

- Commands that display VM/SP control information
- Commands that set and query system features, conditions, and events
- Commands that collect and analyze system information
- 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, DISPLAY, DCP, and DMCP.

The DUMP and DISPLAY commands of CP are privilege class G commands and are used to display control information describing the status of virtual machines.

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

- Virtual program status word (PSW)
- General registers
- Floating-point registers
- Control registers (if your VM/SP directory has the ECMODE option)
- Storage keys
- Virtual storage locations (first-level storage only)

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

- Virtual storage locations (first-level storage only)
- Storage keys
- General registers
- Floating-point registers

- Control registers
- Program status word (PSW)
- Channel address word (CAW)
- Channel status word (CSW)

The DCP and DMCP commands of CP are privilege class C and E commands and are used to display real storage locations. The DMCP command spools the contents of real storage to your virtual printer. The DCP command displays at your terminal the contents of real storage locations.

The class G 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 VM/Interactive Problem Control System Extensions program product can process records written by VMDUMP. For a description of the format and contents of the VMDUMP records, see "VMDUMP Records: Format and Content" in this section.

### *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 buttons on a real computer console. It can also be used to clear storage.

The functions of the SET command are described in detail in the *VM/SP CP Command Reference for General Users*. For debugging, the SET command provides the MSG, WNG, and EMSG operands. These provide messages that may be useful while you are debugging.

The SET MSG function determines whether you receive messages sent by other users via the MSG command. Also, the MSG operand determines whether you receive messages from CP when other users spool reader, printer, or punch files to your virtual machine.

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 the system operator.

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).

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

The QUERY command displays the status of features and conditions set by the SET command for your virtual machine. When you logon, the MSG and WNG

operands of the SET command are set ON; the EMSG operand is set to TEXT; and the SMSG operand is set OFF. To verify these settings, use the QUERY SET command.

## *Commands to Collect and Analyze System Information*

This section discusses six commands to collect and analyze system information when you are debugging. These are the ADSTOP and BEGIN commands and the LOCATE, MONITOR, PER, and TRACE commands.

### 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.

Execution halts when the instruction at the address specified in the ADSTOP command is reached. At this point, you may invoke other CP debugging commands.

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 virtual instruction address must be in real storage at the time the ADSTOP command is issued.

The RANGE keyword of the CP PER command can be used to set multiple address stops. However, unlike the CP ADSTOP command, the program execution halts *after* the execution of the instruction at the given address. 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 20000 executes.

```
PER INSTRUCT RANGE 20000 RANGE 20400
```

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

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

### Locating CP Control Blocks in Storage

Use the LOCATE command to find the address of CP control blocks associated with a particular user, a user's device, or a real system device. The control blocks and their functions are described in the *VM/SP Data Areas and Control Block Logic, Volume 1*.

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 LOCATE and the DCP command to display or the DMCP command to print the control blocks. A discussion of the most important fields of the VMBLOK, VCUBLOK, VDEVBLOK, RCHBLOK, RCUBLOK, and RDEVBLOK are included in "Reading CP Abend Dumps."

## *Commands that Trace Events in Virtual Machines*

Use the TRACE command to trace the following virtual machine events:

- SVC interruption
- I/O interruption
- Program interruption
- External interruption
- Non-I/O privileged instructions
- SIO, SIOF, TIO, CLRIO, HIO, HDV, and TCH instructions.
- Branch instructions
- CCW and CSW instructions

The results collected by the TRACE command are spooled to your virtual printer and to your terminal and/or real printer.

Use the PER command to selectively trace the execution of the instructions that cause specific events. The specific events that can be traced are:

- Successful branches
- The fetching and execution of instructions
- The execution of instructions in the virtual machine that alter storage
- The execution of instructions that alter general purpose registers.

The trace output produced by the PER command can be recorded on the terminal, the virtual printer, or on both the terminal and the printer.

## *Commands that Alter the Contents of Storage*

You can use the STORE, STCP, and ZAP commands to alter the contents of storage.

### Altering the Contents of Virtual Machine Storage

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 (first-level virtual storage only)
- General 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 the Contents of Real Storage

Use the STCP command to alter the contents of real storage. The STCP command cannot alter the real PSW or real registers.

#### Modifying or Dumping CMS MODULE, LOADLIB, or TXTLIB Files

Use the ZAP command to modify or dump MODULE, LOADLIB, or TXTLIB files. ZAP can be used to modify either fixed- or variable-length MODULE files.

ZAP makes use of control records to control processing. These records can be submitted either from the terminal or from a disk file. Using the VER and REP control records, you can verify and replace records or instructions in a control section (CSECT). Using the DUMP control record, you can dump all or part of a CSECT, an entire member of a LOADLIB or TXTLIB file, or an entire module file. For a complete description of the ZAP command, see the *VM/SP Operator's Guide*.

## Debugging CP in a Virtual Machine

Many CP problems can be isolated without standalone 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 on 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 VM/SP directory. See *VM/SP Operating Systems in a Virtual Machine* 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 interruptions
- SVC interruptions

- Program interruptions
- Machine check interruptions
- I/O interruptions
- Free storage requests
- Release of free storage
- Entry into scheduler
- Queue drop
- Run user requests
- Start I/O
- Unstack I/O interruptions
- Storing a virtual CSW
- Test I/O
- Halt Device
- Unstack IOBLOK or TRQBLOK
- NCP BTU (Network Control Program Basic Transmission Unit)
- Spinning on a lock (attached processor or multiprocessor environment)
- SIGP (X'13')
- Clear Channel instruction
- IUCV communications
- SNA console communication services
- DIAGNOSE X'80'
- Start I/O fast release
- Simulated I/O interruptions
- Clear I/O

An installation may optionally specify the size of the CP trace table. To do so, use the SYSCOR macro instruction 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 trace table size or the size specified is smaller than the default size, CP assigns the default size.

For each 256K bytes (or part thereof) of real storage available at IPL time, one page (4096 bytes) is allocated to the CP trace table. Each entry in the CP trace table is 16 bytes long. There are trace table entries for each type of event recorded. The first byte of each trace table entry, the identification code, identifies the type of event being recorded. Figure 67 on page 504 describes the format of each type of trace table entry. The entry shown in Figure 67 for IUCV communications illustrates the general format of an IUCV entry. See the section "IUCV Trace Table Formats" for the formats of trace table entries for each IUCV function, and for a description of each field in the trace table entry.

In addition, some trace table entries are generated by 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 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 trace table is allocated by the main initialization routine, DMKCPI. The first event traced is placed in the lowest trace table address. Each subsequent event is recorded in the next available trace table entry. Once the 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 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 trace table, TRACSTRT, is at location X'0C'. The address of the byte following the end of the trace table, TRACEND, is at location X'10'. And the address of the next available 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 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 trace table, issue the MONITOR STOP command to suppress recording. iref refid=debugm. The pages allocated to the trace table are not released and recording can be restarted at any time by issuing the MONITOR START 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.

| Type of Event | Module | Identification Code (hexadecimal) (See Note 1) | Format of Trace Table Entry |
|---|---|---|---|
| External Interrupt | DMKPSA | 01 | X'01' (0) \| X'0000000000' (1) \| Interrupt Code (6) \| External Old PSW (8–15) |
| SVC Interrupt | DMKSVC | 02 | X'02' (0) \| GR14 or GR15 (See Note 2) (1) \| Instruction Length Code (4) \| Interrupt Code (6) \| SVC Old PSW (8–15) |
| Program Interrupt | DMKPRG | 03 | X'03' (0) \| First 3 Bytes of VMPSW (1) \| Instruction Length Code (4) \| Interrupt Code (6) \| Program Old PSW (8–15) |
| Machine Check Interrupt | DMKMCH | 04 | X'04' (0) \| Address of VMBLOK (1) \| First 4 bytes of 8-byte Interrupt Code (4) \| Machine Check Old PSW (8–15) |
| I/O Interrupt | DMKIOS | 05 | X'05' (0) \| X'00' (1) \| Device Address (2) \| I/O Old PSW + 4 (4) \| CSW (8–15) |
| Free Storage (FREE) | DMKFRE | 06 | X'06' (0) \| Address of VMBLOK (1) \| GR 0 at entry (4) \| GR 1 at exit (8) \| GR 14 (12–15) |
| Return Storage (FRET) | DMKFRE | 07 | X'07' (0) \| Address of VMBLOK (1) \| GR 0 at entry (4) \| GR 1 at entry (8) \| GR 14 (12–15) |
| Enter Scheduler | DMKSCH | 08 | X'08' (0) \| Address of VMBLOK (1) \| Value of VMRSTAT, VMDSTAT, VMOSTAT, and VMQSTAT (4) \| VMQLEVEL VMTLEVEL (8) \| VMIOINT (10) \| VMPEND (12) \| GR 14 (13–15) |
| Queue Drop | DMKSCH | 09 | X'09' (0) \| Address of VMBLOK (1) \| Eligible List Priority VMEPRIOR (4) \| Time Slice End Flag DMKSCHAL (8) \| User Priority VMUPRIOR (9) \| Dispatching Queue Priority VMQPRIOR (10) \| Recent History User CPU Utilization VMUHS (12–15) |
| Run User | DMKDSP | 0A | X'0A' (0) \| X'000000' (1) \| RUNUSER Value from PSA (4) \| RUNPSW Value from PSA (8–15) |
| Start I/O | DMKCNS DMKIOS DMKCPI | 0B | X'0B' (0) \| Condition Code (1) \| Device Address (2) \| Address of IOBLOK (4) \| CAW (8) \| For CC = 1, CSW + 4 otherwise this field is not used (12–15) |
| Unstack I/O Interrupt | DMKDSP | 0C | X'0C' (0) \| X'00' (1) \| Virtual Device Address (2) \| Address of VMBLOK (4) \| Virtual CSW (8–15) |
| Virtual CSW store | DMKVSI | 0D | X'0D' (0) \| Instruction Operation Code (1) \| Virtual Device Address (2) \| Address of VMBLOK (4) \| Virtual CSW (8–15) |
| Test I/O | DMKCPI DMKIOS | 0E | X'0E' (0) \| Condition Code (1) \| Device Address (2) \| Address of IOBLOK (4) \| CAW (8) \| For CC = 1, CSW + 4 otherwise this field is not used (12–15) |
| Halt Device | DMKCNS DMKIOS DMKVSI DMKCPI | 0F | X'0F' (0) \| Condition Code (1) \| Device Address (2) \| Address of IOBLOK (4) \| CAW (8) \| For CC = 1, CSW + 4 otherwise this field is not used (12–15) |

*Notes:*

1. If the installation is running in attached processor mode, the identification code will be OR'd with an X'40' if the activity occurred on the attached processor. If the installation is running ECPS, the identification code is OR'd with an X'80' if the activity occurred in microcode.
2. If the interrupt code (bytes 6 and 7) is 0C, the contents of GR 14 are displayed. For all other interrupt codes, the contents of GR 15 are displayed.
3. Bytes 2 through 15 of a code 11 trace record represent a Basic Transmission Unit, sent or received by a 3704/3705. If CONSYSR/CONEXTR are zero, the BTU was transmitted to the 3704/3705. If they are non-zero, the BTU was received. If CONTCMD equals X'7700', this is an unsolicited BTU response.

**Figure 67 (Part 1 of 2). CP Trace Table Entries**

| Type of Event | Module | Identification Code (hexadecimal) (See Note 1) | Format of Trace Table Entry |
|---|---|---|---|
| Unstack IOBLOK or TRQBLOK | DMKDSP | 10 | X'10' (0) \| Address of VMBLOK (1) \| Value of VMRSTAT, VMDSTAT, VMOSTAT, and VMQSTAT (4) \| Address of IOBLOK or TRQBLOK (8) \| Interrupt Return Address (12–15) |
| NCP BTU (See Note 3) | DMKRNH | 11 | X'11' (0) \| X'00' (1) \| CONSRID (2) \| CONDEST (4) \| CONRTAG (6) \| CONSYSR CONEXTR (8) \| CONTCMD (10) \| CONFUNC CONDFLG (12) \| CONDCNT (14–15) |
| Spinning on lock | DMKLOK | 12 | X'12' (0) \| Address of VMBLOK (1) \| Lockword Address (4) \| Return Address (8) \| Lockword Contents (12–15) |
| SIGP issued | DMKEXT | 13 | X'13' (0) \| Return Address (1) \| 00 (4) \| Condition Code (5) \| Real Processor Address (6) \| Function Code (8) \| Order Code (10) \| Status of Condition Code = 1 (12–15) |
| Clear Channel issued | DMKVSI | 14 | X'14' (0) \| Condition Code (1) \| Device Address (2) \| Address of VMBLOK (4) \| Virtual CSW (8–15) |
| IUCV Communication | DMKIUA | 15 | X'15' (0) \| Function Code (1) \| Path id or Unused (2) \| Address of IUCVBLOK (4) \| Usage varies by function code (See "IUCV Trace Table Formats" for details) (8) \| Address of IUCV Instruction (13–15) |
| SNA CCS | DMKVCV | 16 | X'16' (0) \| Transaction Type (1) \| See "SNA CCS Entries in CP Internal Trace Table" for details. (2–15) |
| DIAGNOSE X'80' | DMKMHC | 17 | X'17' (0) \| Condition Code (1) \| X'0000' (2) \| HCBLOK Address (4) \| MSSF Command Word (8) \| MSFBLOK Address (12–15) |
| Start I/O Fast Release | DMKIOS | 18 | X'18' (0) \| Condition Code (1) \| Device Address (2) \| Address of IOBLOK (4) \| CAW (8) \| For CC = 1, CSW + 4 otherwise this field is not used (12–15) |
| Simulated I/O Interrupt | DMKIOT | 19 | X'19' (0) \| X'00' (1) \| Device Address (2) \| Address of either DMKDID or DMKACR in GR 12 at entry (4) \| CSW (8–15) |
| | | 1A | Reserved for HPO use. |
| CLEAR I/O | DMKIOS | 1B | X'1B' (0) \| Condition Code (1) \| Device Address (2) \| Address of IOBLOK (4) \| CAW (8) \| For CC = 1, CSW + 4 otherwise this field is not used. (12–15) |

*Notes:*

1. If the installation is running in attached processor mode, the identification code will be OR'd with an X'40' if the activity occurred on the attached processor. If the installation is running ECPS, the identification code is OR'd with an X'80' if the activity occurred in microcode.
2. If the interrupt code (bytes 6 and 7) is 0C, the contents of GR 14 are displayed. For all other interrupt codes, the contents of GR 15 are displayed.
3. Bytes 2 through 15 of a code 11 trace record represent a Basic Transmission Unit, sent or received by a 3704/3705. If CONSYSR/CONEXTR are zero, the BTU was transmitted to the 3704/3705. If they are non-zero, the BTU was received. If CONTCMD equals X'7700', this is an unsolicited BTU response.

**Figure 67 (Part 2 of 2). CP Trace Table Entries**

## Abend Dumps

There are three kinds of abnormal termination dumps possible when using CP. If the problem program cannot continue, it terminates and in some cases attempts to issue a dump. Likewise, if the operating system for your virtual machine cannot continue, it terminates and, in some cases, attempts to issue a dump. In the VM/SP 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. The CP abnormal termination dumps can be directed to a printer or tape or be dynamically allocated to DASD. If the dump is directed to a tape, the dumped data must fit on one reel of tape. Multiple tape volumes are not supported by VM/SP. The historical data on the tape is in print line format and can be processed by user-created programs or via CMS commands. Specify the output device for CP abend dumps with the CP SET DUMP command. Refer the *VM/SP Operator's Guide* for the format of the SET DUMP command.

### *How to Print a CP Abend Dump from Tape*

When the CP abend dump is sent to a tape, the records are 131 characters long, unblocked, and contain carriage control characters.

To print the tape, first make sure the tape drive is attached to your system. Next, define the printer and tape file.

```
FILEDEF ddname1 PRINTER (RECFM FM LRECL 131)

FILEDEF ddname2 {TAP2} (DEN 1600 RECFM U LRECL 132)
                {TAP1}
```

Then use the MOVEFILE command to print the tape:

```
MOVEFILE ddname2 ddname1
```

## 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. When the dump is directed to a direct access device, IPCS (Interactive Problom Control Service) must be used to format and print the dump. IPCS commands format and print:

- Control blocks
- General registers
- Floating-point registers
- Control registers
- TOD (Time-of-Day) Clock
- Processor Timer
- Storage
- If in attached processor or multiprocessor 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 tape or the 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.

When the Control Program 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 the Control Program. "Part 1: Control Program (CP)" contains 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 Data Areas and Control Block Logic, Volume 1* for a description of the CP control blocks. Figure 68 on page 512 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* for obtaining the original copy of the CP load map.

### *Reason for the Abend*

Determine the immediate reason for the abend. You need to examine several fields in the PSA (Prefix Storage Area), to find the reason for the abend. In a uniprocessor system, the PSA is in locations 0 to 4095. In an attached processor or multiprocessor system, each processor has its own PSA in addition to the absolute PSA in locations 0 to 4095.

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 a Pageable Module." Figure 74 on page 542 in "Appendix A: System/370 Information" 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 a Pageable Module". Figure 74 on page 542 in Appendix A describes the format of an Extended Control PSW.

3. Examine the old PSW 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 74 on page 542 in Appendix A 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 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 Data Areas and Control Block Logic, Volume 1.*

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

1. CP Running Status Field

   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 67 on page 504 describes the format of each type of trace table entry.

**Note:** If the system was in attached processor or multiprocessor mode, the trace table pointers are in absolute page zero.

## *Register Usage*

In order to trace control blocks and modules, it is necessary to know the CP register usage conventions.

The 16 general registers have many uses that vary depending upon the operation. The following table shows the use of some of the general registers.

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

The following general registers usually contain the same information.

| Register | Contents |
|---|---|
| GR 11 | The address of the active VMBLOK. |
| GR 12 | The base register for the module executing. |
| GR 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*

There are three save areas that may be helpful in debugging CP. If a module was called by an SVC, examine the SAVEAREA storage area. SAVEAREA is not in the PSA; the address of the SAVEAREA is found in general register 13. If a module was called by a branch and link, the general registers are saved in the PSA in an area called BALRSAVE (X'240'). The DMKFRE save area and work area is also in the PSA: these areas are used only by the DMKFREE and DMKFRET routines. The DMKFRE save area (FREESAVE) is at location X'280' and its work area (FREEWORK) follows at location X'2C0'.

Save areas used by attached processor and multiprocessor support are DUMPSAVE, SIGSAVE, LOKSAVE, MFASAVE, SWTHSAVE, LOCKSAVE, and SVCREGS. These save areas are all in the PSA. All except LOCKSAVE and SVCREGS are 16 words 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 trace backwards again.

2. BALRSAVE

   All the general 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 registers are saved in FREESAVE before DMKFRE executes. Use this address to trace module control backwards.

| Field | Contents |
|---|---|
| FREER15 | The entry point (DMKFREE or DMKFRET). |
| 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 calls to DMKFRET). |
| FREER0 | Contains the number of doublewords requested or returned. |

4. DUMPSAVE

   All the general 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 GR0 through GR15. GR12 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 multiprocessor/attached processor-only module that handles all signaling requests. When a signal request is issued, DMKEXTSP is called. On entry, DMKEXTSP stores GR12 through GR15, and GR0 through GR6. GR7 through GR11 are not saved. The remainder of SIGSAVE is used as a work area. GR14 contains the caller's return address.

6. LOKSAVE

   All the general registers are stored in LOKSAVE (displacement X'580') before DMKLOK executes. DMKLOK is a multiprocessor/attached processor-only module that manipulates certain locks. The registers are stored in the order GR0 through GR15. GR14 contains the caller's return address.

7. MFASAVE

   All the general registers are stored in MFASAVE (displacement X'5C0') before DMKMCTMA executes. DMKMCTMA is the entry into DMKMCT, a multiprocessor/attached processor-only module, that handles malfunction alert interrupts. The registers are stored space in the order GR0 through GR15. GR14 and GR15 contain the caller's return address.

8. SWTHSAVE

   All the general registers are stored in SWTHSAVE (displacement X'600') by DMKSTK and DMKVMASW. DMKVMASW is an entry that is used only in multiprocessor/attached processor systems to switch a user's page table pointers. The registers are stored in the order GR0 through GR15. GR14 contains the caller's return address. All entries to DMKSTK store registers GR0 through GR15 in SWTHSAVE.

9. LOCKSAVE

LOCKSAVE (displacement X'640') is a four-word save area used by the LOCK macro to save GR14, GR15, GR0, and GR1 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 GR12 through GR15 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 68 on page 512 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 Data Areas and Control Block Logic, Volume 1*.

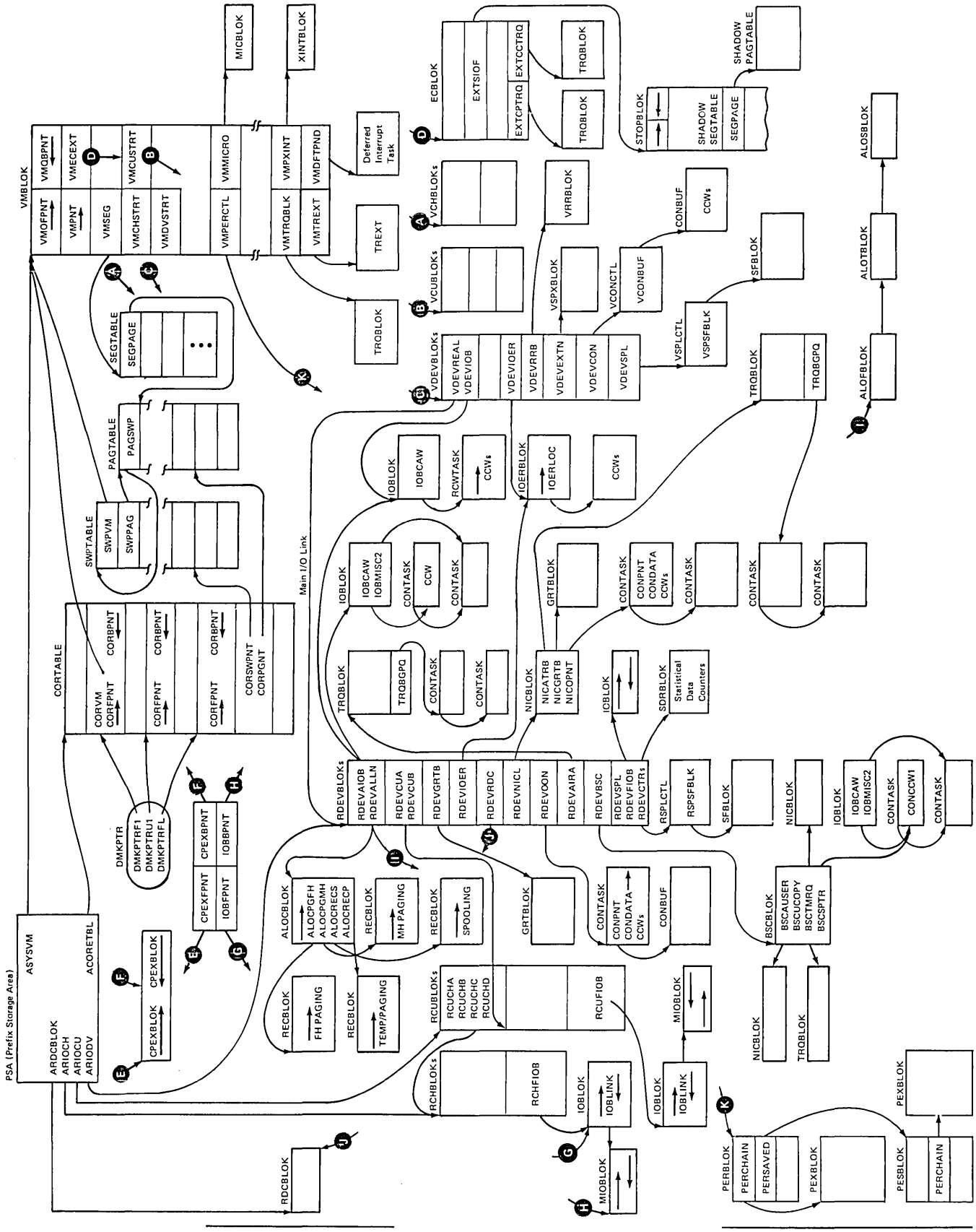



**Figure 68. CP Control Block Relationships**

### VMBLOK

The address of the VMBLOK is in general register 11.

Examine the following VMBLOK fields:

1. The virtual machine running status is contained in VMRSTAT (displacement X'58'). The value of this field indicates the running status.

2. The virtual machine dispatching status is contained in VMDSTAT (displacement X'59'). The value of this field indicates the dispatching status.

3. Examine the virtual PSW and the last virtual machine privileged instruction. The virtual machine PSW is saved in VMPSW (displacement X'A8') and the virtual machine privileged or tracing instruction is saved in VMINST (displacement X'98').

4. Find the name of the last CP command that executed in VMCOMND (displacement X'148').

5. Check the status of I/O activity. The following fields contain pertinent information.

   a. VMPEND (displacement X'63') contains the interrupt pending summary flag. The value of VMPEND identifies the type of interrupt.

   b. VMFSTAT (displacement X'68') contains the virtual machine features.

   c. VMIOINT (displacement X'6A') contains the I/O interrupt pending flag. Each bit represents a channel (0 through 15). An interrupt pending is indicated by a 1 in the corresponding bit position.

   d. VMIOACTV (displacement X'36') is the active channel mask. An active channel 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 register 6 contains the address of the active VCHBLOK. Examine the following fields:

1. The virtual channel address is contained in VCHADD (displacement X'00').

2. The status of the virtual channel is found in the VCHSTAT field (displacement X'06'). The value of this field indicates the virtual channel status.

3. The value of the VCHTYPE field (displacement X'07') indicates the virtual channel type.

### VCUBLOK

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

1. The virtual control unit address is found in the VCUADD field (displacement X'00').

2. The value of the VCUSTAT field (displacement X'06') indicates the status of the virtual control unit.

3. The value of the VCUTYPE field (displacement X'07') indicates 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 register 8 contains the address of the active VDEVBLOK. Useful information is contained in the following fields:

1. The virtual device address is found in the VDEVADD field (displacement X'00').

2. The value of the VDEVSTAT field (displacement X'06') describes the status of the virtual device.

3. The value of the VDEVFLAG field (displacement X'07') indicates the device-dependent information.

4. The VDEVCSW field (displacement X'08') contains the virtual channel status word for the last interrupt.

5. The VDEVREAL field (displacement X'24') contains the pointer to the real device block, RDEVBLOK.

6. The VDEVIOB field (displacement X'34') contains the pointer to the active IOBLOK.

7. For console devices, the value of the VDEVCFLG field (displacement X'26') describes the virtual console flags.

8. For spooling devices, the value of the VDEVSFLG field (displacement X'27') describes the virtual spooling flags.

9. For output spooling devices, the VDEVEXTN field (displacement X'10') contains the pointer to the virtual spool extension block, VSPXBLOK.

10. The value of the VDEVFLG2 field (displacement X'38') describes the Reserve/Release flags and other miscellaneous conditions.

11. For Reserve/Release minidisks, VDEVRRB (displacement X'3C') contains the address of the VRRBLOK.

**RCHBLOK**

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

1. The real channel address is found in the RCHADD field (displacement X'00').

2. The value of the RCHSTAT field (displacement X'04') describes the status of the real channel.

3. The value of the RCHTYPE field (displacement X'05') describes the real channel type.

4. The RCHFIOB field (displacement X'08') is the pointer to the first IOBLOK in the queue and the RCHLIOB field (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 register 7 points to the current RCUBLOK. Examine the following fields:

1. The RCUADD field (displacement X'00') contains the real control unit address.

2. The value of the RCUSTAT field (displacement X'04') describes the status of the control unit.

3. RCUCHA (displacement X'10') points to the Primary RCHBLOK.

4. RCUCHB (displacement X'14') points to the first alternate RCHBLOK.

5. RCUCHC (displacement X'18') points to the second alternate RCHBLOK.

6. RCUCHD (displacement X'1C') points to the third alternate RCHBLOK.

7. The value of the RCUTYPE field (displacement X'05') describes the type of the real control unit.

8. The RCUFIOB field (displacement X'08') points to the first IOBLOK in the queue and the RCULIOB field (displacement X'0C') points to the last IOBLOK in the queue.

**RDEVBLOK**

The address of the first RDEVBLOK is found in the ARIODV field (displacement X'3BC') of the PSA. General 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. The RDEVADD field (displacement X'00') contains the real device address.

2. The values of the RDEVSTAT (displacement X'04'), RDEVSTA2 (displacement X'45'), and RDEVSTA4 fields describe the status of the real device.

3. The value of the RDEVFLAG field (displacement X'05') indicates device flags. These flags are device-dependent.

4. The value of the RDEVTYPC field (displacement X'06') describes the device type class and the value of the RDEVTYPE field (displacement X'07') describes the device type. Refer to Figure 69 on page 517 for the list of possible device type class and device type values.

5. The RDEVAIOB field (displacement X'24') contains the address of the active IOBLOK.

6. The RDEVUSER field (displacement X'28') points to the VMBLOK for a dedicated user.

7. The RDEVATT field (displacement X'2C') contains the attached virtual address.

8. The RDEVIOER field (displacement X'48') contains the address of the IOERBLOK for the last CP error.

9. For spooling unit record devices, the RDEVSPL field (displacement X'18') points to the active RSPLCTL block.

10. For real 370X Communications Controllers, several pointer fields are defined. The RDEVEPDV field (displacement X'1C') points to the start of the free RDEVBLOK list for EP lines. The RDEVNICL field (displacement X'38') points to the network control list and the RDEVCKPT field (displacement X'3C') points to the CKPBLOK for re-enable. Also, the RDEVMAX field (displacement X'2E') is the highest valid NCP resource name and the RDEVNCP field (displacement X'30') is the reference name of the active 3705 NCP.

11. For terminal devices, additional flags are defined. The value of the RDEVTFLG field (displacement X'3A') describes the additional flags.

12. For terminals, an additional flag is defined. The value of the RDEVTMCD field (displacement X'34') describes the line code translation to be used.

DEVICE CLASS CODES

| Code | Device Class |
|---|---|
| X'80' | Terminal Device |
| X'40' | Graphics Device |
| X'20' | Unit Record Input Device |
| X'10' | Unit Record Output Device |
| X'08' | Magnetic Tape Device |
| X'04' | Direct Access Storage Device |
| X'02' | Special Device |
| X'01' | Fixed-Block Storage |

DEVICE TYPE CODES

- For Terminal Device Class

| Code | Device Type |
|---|---|
| X'80' | Binary Synchronous Line for Remote |
| X'40' | 2700 Binary Synchronous Line |
| X'40' | 2955 Communication Line |
| X'20' | Telegraph Terminal Control Type II |
| X'20' | Teletype Terminal |
| X'1C' | Undefined Terminal Device |
| X'18' | IBM 2741 Communication Terminal |
| X'18' | IBM 3767 Communication Terminal |
| X'14' | IBM 1050 Data Communication System |
| X'10' | IBM Terminal Control Type I |
| X'01' | Dial Feature |
| X'00' | IBM 3210 Console |
| X'00' | IBM 3215 Console |
| X'00' | IBM 2150 Console |
| X'00' | IBM 1052 Console |

**Figure 69 (Part 1 of 5). CP Device Classes, Types, Models, and Features**

- For Graphics Device Class

| Code | Device Type |
|---|---|
| X'80' | IBM 2250 Display Unit |
| X'40' | IBM 2260 Display Station |
| X'20' | IBM 2265 Display Station |
| X'10' | IBM 3066 Console |
| X'08' | IBM 1053 Printer |
| X'04' | IBM 3138 System Console |
| X'04' | IBM 3148 System Console |
| X'04' | IBM 3158 System Console |
| X'04' | IBM 3277 Display Station |
| X'01' | IBM 3278 Display Station |
| X'01' | IBM 3279 Display Station |
| X'02' | IBM 3230 Printer |
| X'02' | IBM 3268 Printer |
| X'02' | IBM 3284 Printer |
| X'02' | IBM 3286 Printer |
| X'02' | IBM 3287 Printer |
| X'02' | IBM 3288 Printer |
| X'02' | IBM 3289 Printer |
| X'02' | IBM 4250 Printer |

- For Unit Record Input Device Class

| Code | Device Type |
|---|---|
| X'90' | IBM 2520 Card Reader/Punch |
| X'88' | IBM 1442 Card Reader/Punch |
| X'84' | IBM 3505 Card Reader |
| X'82' | IBM 2540 Card Reader |
| X'81' | IBM 2501 Card Reader |
| X'80' | Card Reader |
| X'40' | Timer |
| X'24' | IBM 1017 Paper Tape Reader |
| X'22' | IBM 2671 Paper Tape Reader |
| X'21' | IBM 2495 Magnetic Tape Cartridge Reader |
| X'20' | Tape Reader |

**Figure 69 (Part 2 of 5). CP Device Classes, Types, Models, and Features**

- For Unit Record Output Device Class

| Code | Device Type |
|---|---|
| X'90' | IBM 2520 Card Punch |
| X'88' | IBM 1442 Card Punch |
| X'84' | IBM 3525 Card Punch |
| X'82' | IBM 2540 Card Punch |
| X'80' | Card Punch |
| X'4A' | IBM 4245 Printer |
| X'47' | IBM 3262 Printer |
| X'46' | IBM 3289 Printer |
| X'45' | IBM 3800 Printing Subsystem |
| X'44' | IBM 1443 Printer |
| X'43' | IBM 3203 Printer |
| X'42' | IBM 3211 Printer |
| X'41' | IBM 1403 Printer |
| X'40' | Printer |
| X'24' | IBM 1018 Paper Tape Punch |
| X'20' | Tape Punch |

- For Magnetic Tape Device Class

| Code | Device Tape |
|---|---|
| X'80' | IBM 2401 Tape Drive |
| X'40' | IBM 2415 Tape Drive |
| X'20' | IBM 2420 Tape Drive |
| X'10' | IBM 3420 Tape Drive |
| X'08' | IBM 3410/3411 Tape Drive |
| X'04' | IBM 8809 Tape Drive |
| X'02' | IBM 3430 Tape Drive |

- For Direct Access Storage Device Class

| Code | Device Type |
|---|---|
| X'80' | IBM 2301 Parallel Drum |
| X'80' | IBM 2303 Serial Drum |
| X'80' | IBM 2311 Disk Storage Drive |
| X'80' | IBM 2321 Data Cell Drive |
| X'40' | IBM 2314 Disk Storage Facility |
| X'40' | IBM 2319 Disk Storage Facility |
| X'20' | IBM 3380 Disk Storage Facility |
| X'10' | IBM 3330 Disk Storage Facility |
| X'10' | IBM 3333 Disk Storage and Control |
| X'08' | IBM 3350 Disk Storage Facility |
| X'04' | IBM 3375 Disk Storage Facility |
| X'02' | IBM 2305 Fixed Head Storage Device |
| X'01' | IBM 3340 Disk Storage Facility |

**Figure 69 (Part 3 of 5). CP Device Classes, Types, Models, and Features**

- For Special Device Class

| Code | Device Type |
|---|---|
| X'80' | Channel-to-Channel Device (CTCA or 3088) |
| X'40' | 370X Programmable Communications Controller |
| X'20' | 3851 Mass Storage Controller |
| X'04' | SRF (7443) device |
| X'01' | Device unsupported by VM/SP |

For Fixed-Block Storage Device Class

| Code | Device Type |
|---|---|
| X'02' | 3370 |
| X'01' | 3310 |
| X'00' | Generic Fixed-Block (see Note) |

**Note:**
Code X'00' applies to a device whose specific type CP has not yet determined. The proper bit value is assigned when a 'Read Device Characteristics' command is issued at IPL.

MODEL CODES (Column 35 in Accounting Card)

As specified in the RDEVICE macro at system generation.

FEATURE CODES (Column 36 in Accounting Card)

- For Printer Devices

| Code | Feature |
|---|---|
| X'01' | UCS |

**Figure 69 (Part 4 of 5). CP Device Classes, Types, Models, and Features**

- For Magnetic Tape Devices

| Code | Feature |
|---|---|
| X'80' | 7 Track |
| X'40' | Dual Density |
| X'20' | Translate |
| X'10' | Data Conversion |

- For Direct Access Storage Devices

| Code | Feature |
|---|---|
| X'80' | Rotational Position Sensing (RPS) |
| X'40' | Extended Sense Bytes (24 bytes) |
| X'20' | Top Half of 2314 Used as 2311 |
| X'10' | Bottom Half of 2314 Used as 2311 |
| X'08' | 35MB Data Module (mounted) |
| X'04' | 70MB Data Module (mounted) |
| X'02' | Reserve/Release are valid CCW operation codes |
| X'01' | 3330V Virtual MSS volume |

- For special devices

| Code | Feature |
|---|---|
| X'20' | Type II channel adapter for 370X |
| X'10' | Type I channel adapter for 370X |

**Figure 69 (Part 5 of 5). CP Device Classes, Types, Models, and Features**

### *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.

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.

### *VMDUMP Records: Format and Content*

When a user issues the VMDUMP command, CP dumps virtual storage of the user's virtual machine. CP stores 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. -- Fig 'f11' unknown -- 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) followed by a TAG area followed by dump information. All other spool file records contain only a SPLINK followed by dump information.

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

| hexadecimal offset | length | content |
|---|---|---|
| 0 | 4 bytes | the DASD location (DCHR) of the next page buffer |
| 4 | 4 bytes | the DASD location (DCHR) of the previous page buffer |
| 8 | 4 bytes | binary zeros |
| C | 4 bytes | the number of data records in the buffer |

The TAG area contains either binary zeros 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 zeros.

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 Data Areas and Control Block Logic, Volume 1*. For a description of DMPKYREC1 and DMPKYREC2, see DMPKYREC also in *VM/SP Data Areas and Control Block Logic, Volume 1*.

### Locating Logical Dump Records

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

$$loc = \frac{240+16n+4096n}{4096}$$

*where:*

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.

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:

$$loc = \frac{240+(16x3)+(4096x3)}{4096}$$

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

quotient = 3

remainder = 288

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

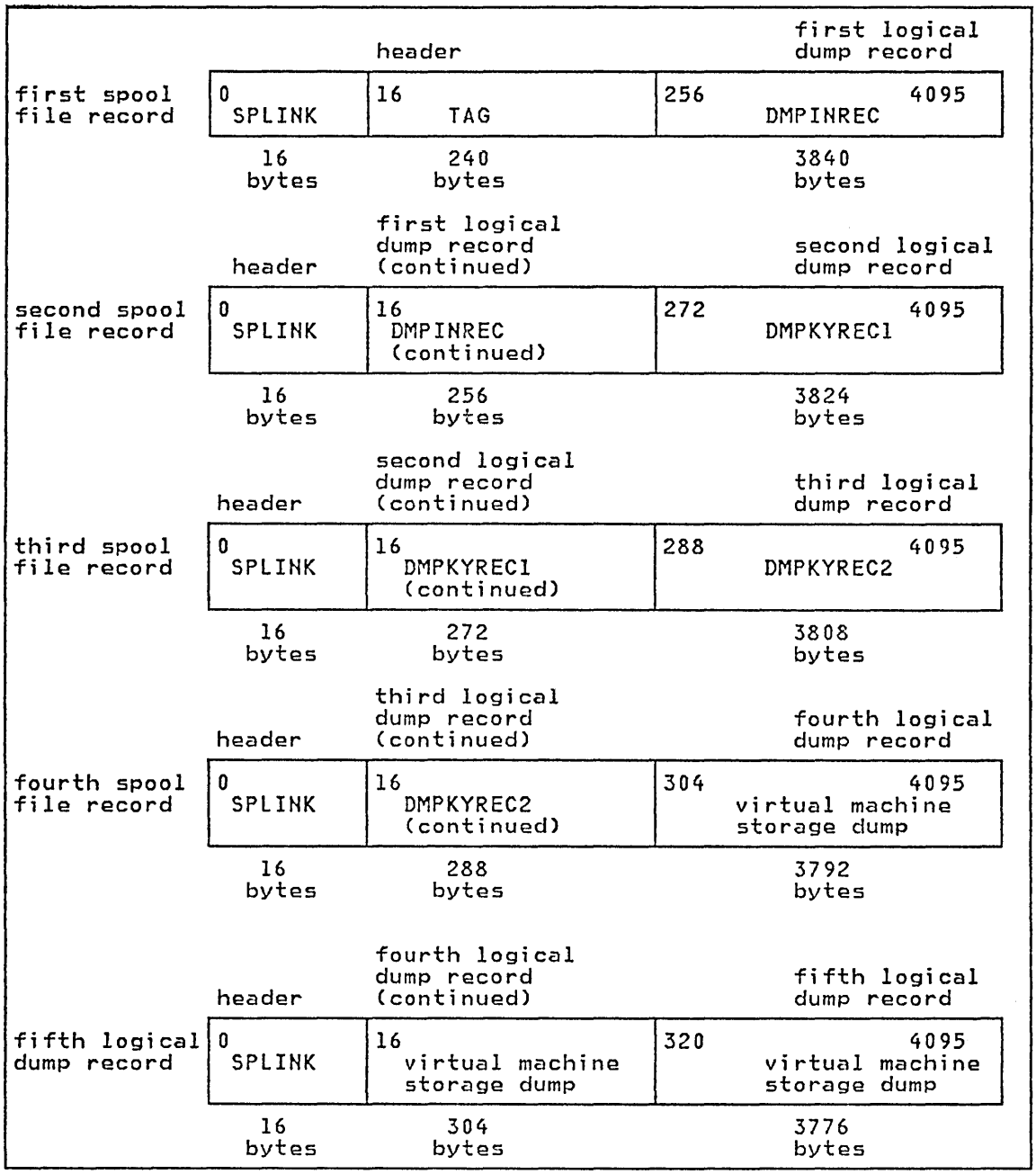



**Figure 70. VMDUMP Record Format**

# Debugging With CMS

This section describes the debug tools that CMS provides. These tools can be used to help you debug CMS or a problem program. In addition, a CMS user can use the CP commands to debug. Information that is often useful in debugging is also included. The following topics are discussed in this section:

- CMS debugging commands
- Load maps
- Reading CMS dumps
- Control block summary

## CMS Debugging Commands

CMS provides two commands that are useful in debugging: DEBUG and SVCTRACE. Both commands execute from the terminal.

The debug environment is entered whenever:

- The DEBUG command is issued
- A breakpoint is reached
- An external or program interrupt occurs

CMS will not accept other commands while in the debug environment. However, while in the debug environment, subcommands of the DEBUG command can be used to:

- Set breakpoints (address stops) that stop program execution at specific locations.
- Display the contents of the CAW (channel address word), CSW (channel status word), old PSW (program status word), or general registers at the terminal.
- Change the contents of the control words (CAW, CSW, and PSW) and general registers.
- Dump all or part of virtual storage at the printer.
- Display the contents of up to 56 bytes of virtual storage at the terminal.
- Store data in virtual storage locations.
- Allow an origin or base address to be specified for the program.
- Assign symbolic names to specific storage locations.
- Close all open files and I/O devices and update the master file directory.
- Exit from the debug environment.

The SVCTRACE command records information for all SVC calls. When the trace is terminated, the information recorded up to that point is printed at the system printer.

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

## *DEBUG*

The DEBUG command provides support for debugging programs at a terminal. The virtual machine operator can stop the program at a specified location in order to examine and alter virtual storage, registers, and various control words. Once CMS is in the debug environment, the virtual machine operator can issue the various DEBUG subcommands. However, in the debug environment, all of the other CMS commands are considered invalid.

Any DEBUG subcommand may be entered if CMS is in the debug environment and the keyboard is unlocked. The following rules apply to DEBUG subcommands:

1. No operand should be longer than eight characters. All operands longer than eight characters are left-justified and truncated on the right after the eighth character.

2. The DEFINE subcommand must be used to create all entries in the DEBUG symbol table.

3. The DEBUG subcommands can be truncated. The following is a list of all valid DEBUG subcommands and their minimum truncation.

| Subcommand | Minimum Truncation |
|---|---|
| BREAK | BR |
| CAW | CAW |
| CSW | CSW |
| DEFINE | DEF |
| DUMP | DU |
| GO | GO |
| GPR | GPR |
| HX | HX |
| ORIGIN | OR |
| PSW | PSW |
| RETURN | RET |
| SET | SET |
| STORE | ST |
| X | X |

One way to enter the debug environment is to issue the DEBUG command. The message

```
DMSDBG728I DEBUG ENTERED
```

appears at the terminal. Any of the DEBUG subcommands may be entered. To continue normal processing, issue the RETURN subcommand. Whenever a program check occurs, the DMSABN routine gains control. Issue the DEBUG command at this time if you wish CMS to enter the debug environment.

Whenever a breakpoint is encountered, a program check occurs. The message

```
DMSDBG728I DEBUG ENTERED
BREAKPOINT YY AT XXXXX
```

appears on the terminal. Follow the same procedure to enter subcommands and resume processing as with a regular program check.

An external interrupt, which occurs when the CP EXTERNAL command is issued, causes CMS to enter the debug environment. The message

```
DMSDBG728I DEBUG ENTERED
EXTERNAL INTERRUPT
```

appears on the console. Any of the DEBUG subcommands may be issued. To exit from the debug environment after an external interrupt, use GO.

While CMS is in the debug environment, the control words and low storage locations contain the debug program values. The debug program saves the control words and low storage contents (X'00'. through X'100'.) of the interrupted routine at location X'C0'..

## *&CRASH*

The &CRASH command is used as an aid in debugging the EXEC 2 interpreter DMSEXE and is intended to be used by system support people only. It is generally only useful when used in conjunction with a current listing of module DMSEXE.

**Note:** The &CRASH command is *not* used for debugging programs or EXEC files written in the EXEC 2 language. For information on debugging programs and EXECs written in the EXEC 2 language, see the &TRACE command in *VM/SP EXEC 2 Reference.*

The format of the &CRASH command is:

&CRASH [text]

*where:*

text — if specified, is the character string contained in memory just prior to the instruction that caused the &CRASH command to be executed.

**WARNING:** Unless this command is used as described, abnormal termination of CMS and loss of data may occur.

Usage Notes:

1. &CRASH should be used only after the CP TRACE PROG command is issued.

2. Execution of the &CRASH command causes entry to CP command mode. One of the following statements should be issued to continue execution:

   BEGIN intaddr+2 — to continue execution

   *where:*

   intaddr+2 — is the address of the interrupt plus two

-or-

BEGIN r14addr — to terminate the EXEC file with a return code as given in register 15

*where:*

r14addr — is the address in register 14

Simply issuing the BEGIN command causes abnormal termination of CMS.

3. The registers contain the following information when CP command mode is entered:

   R1 = address of the unsubstituted &CRASH arguments.

   R10, R11 = main base registers (the address of the label "MAIN" and "MAIN+4096")

   R12 = the address of the label "EXEC"

   R13 = the address of DSECT "AREA"

   R14 = the address of where to resume if termination is needed

   R15 = 0

   If execution is resumed at the next instruction in memory, the EXEC file will continue execution at the next EXEC statement.

   If execution is resumed at the location addressed by register 14, the EXEC file will terminate and yield a return code given by the value of register 15.

Example of use:

```
.
.
.
&ERROR &IF &RC < 0 &IF &LINENUM > 50 &CRASH UNKNOWN COMMAND
.
.
.
```

This statement causes the &CRASH command to execute when a command error occurs that returns a return code less than zero and on a line above line 50. If the &CRASH command is executed, the results might look like this:

```
*** 2CF158 PROG  0001 ==> 01F1C8

D G                <<<-- Show all the registers
GPR  0 =  FFFFD798  001BAE48  00000006  00000038
GPR  4 =  00006885  001BAF57  002CC8EC  002CCF8C
GPR  8 =  001BAF57  001BAF5E  002CC8E2  002CD8E2
GPR 12 =  002CC690  001BACE8  002CF162  00000000

D T1BAE48.20       <<<-- Display the &CRASH arguments
1BAE40   E20F50C3 D9C1E2C8 4050D3C9 D5C5D5E4  *S.&CRASH UNKNOWN*
1BAE50   D44000C5 40C1D3D3 40004040 40404040  * COMMAND sse.<> *
1BAE60   40404040 40404040 40404040 40404040  *                *

D T1BACE8.10       <<<-- Display which EXEC file was interrupted
1BACE0   00011618 12E81BFF C1C2C3C3 40404040  *........ABCD    *
1BACF0   40404040 C5E9C5C3 40404040 40404040  *    EXEC        *
```

```
D T2CF158.10      <<<-- Display where the interrupt happened
2CF150   C01858C0 D0F81BFF 000007F6 50C3D9C1  *.....8.....6&CRA*
2CF160   E2C8189F 47F0C1E2 45E0AAB8 078745E0  *SH...0AS........*

B 2CF15A          <<<-- Continue as if no interrupt happened
```

## Nucleus Load Map

Each time the CMS resident nucleus is loaded on a DASD and an IPL can be performed on that DASD, a 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 "Generating a CMS Nucleus" in the *VM/SP Installation Guide.*

## Load Map

The 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. When using the Debug environment to analyze a program, use the program's load map to help in displaying information.

There are two ways to get a load map.

1. 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.

2. 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 |
|---|---|

*where:*

filename is the module whose map is to be displayed. The filetype must be MODULE.

## Reading CMS Abend Dumps

If an abend dump is desired when CMS abnormally terminates, the terminal operator must enter the DEBUG command and then the DUMP subcommand. The dump formats and prints:

- General registers

- 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. In order to find the cause of a CMS problem, you must be familiar with the structure and functions of CMS. Refer to "Part 2: Conversational Monitor System (CMS)" for functional information. The following discussion on reading CMS dumps refers to several CMS control blocks and fields in the control blocks. Refer to the *VM/SP Data Areas and Control Block Logic, Volume 2* for details on CMS control blocks. Figure 71 on page 531 shows the CMS control block relationships. You also need a current CMS nucleus load map in order to analyze the dump.

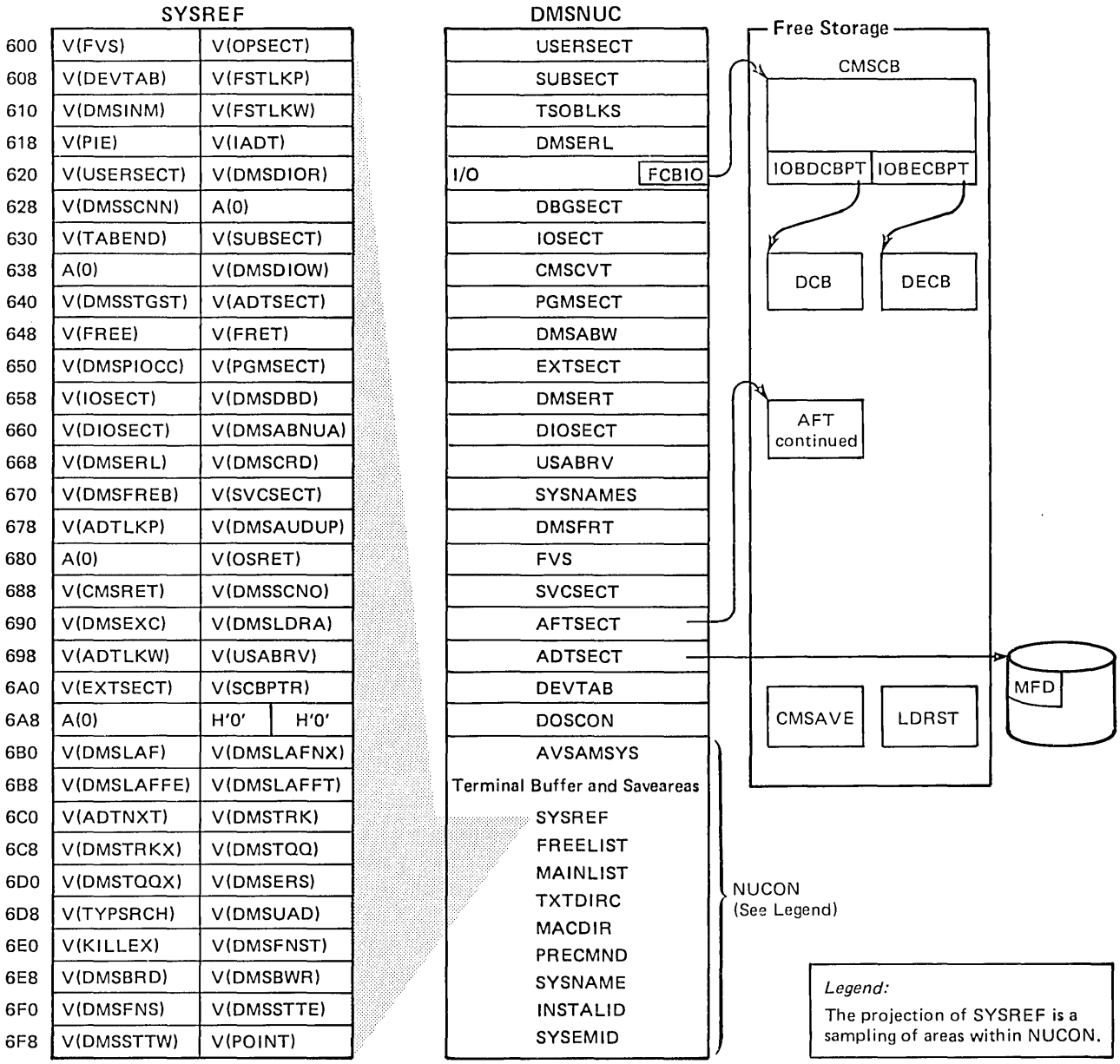



Figure 71. CMS Control Blocks

### *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. The *VM/SP System Messages and Codes* manual 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 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 74 on page 541 in "Appendix A: System/370 Information" 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 the *VM/SP Data Areas and Control Block Logic, Volume 2*. 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 registers.

| Field | Location | Contents |
|---|---|---|
| FPRLOG | X'160' | User floating-point registers |
| GPRLOG | X'180' | User general 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

  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 CVT (Communications Vector Table)

  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 ADT (Active Disk Table), or AFT (Active File Table) in NUCON.

See a CMS nucleus map for the location of these CSECTs.

## *Register Usage*

In order to trace control blocks and modules, it is important to know the CMS register usage conventions.

| Register | Contents |
|---|---|
| GR1 | Address of the PLIST |
| GR12 | Program's entry point |
| GR13 | Address of a 12-doubleword work area for an SVC call |
| GR14 | Return address |
| GR15 | 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 71 on page 531 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 attempt 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 DMSSAR 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.

# Appendixes

- Appendix A: System/370 Information
- Appendix B: VM Monitor Tape Format and Content
- Appendix C: CMS Macro Library

# Appendix A. System/370 Information

## Control Registers

The control registers are used to maintain and manipulate control information that resides outside the 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 72 is a summary of the control register allocation and Figure 73 on page 538 lists the facility associated with each control register.

Figure 74 on page 541 is a description of the EC (Extended Control) PSW.

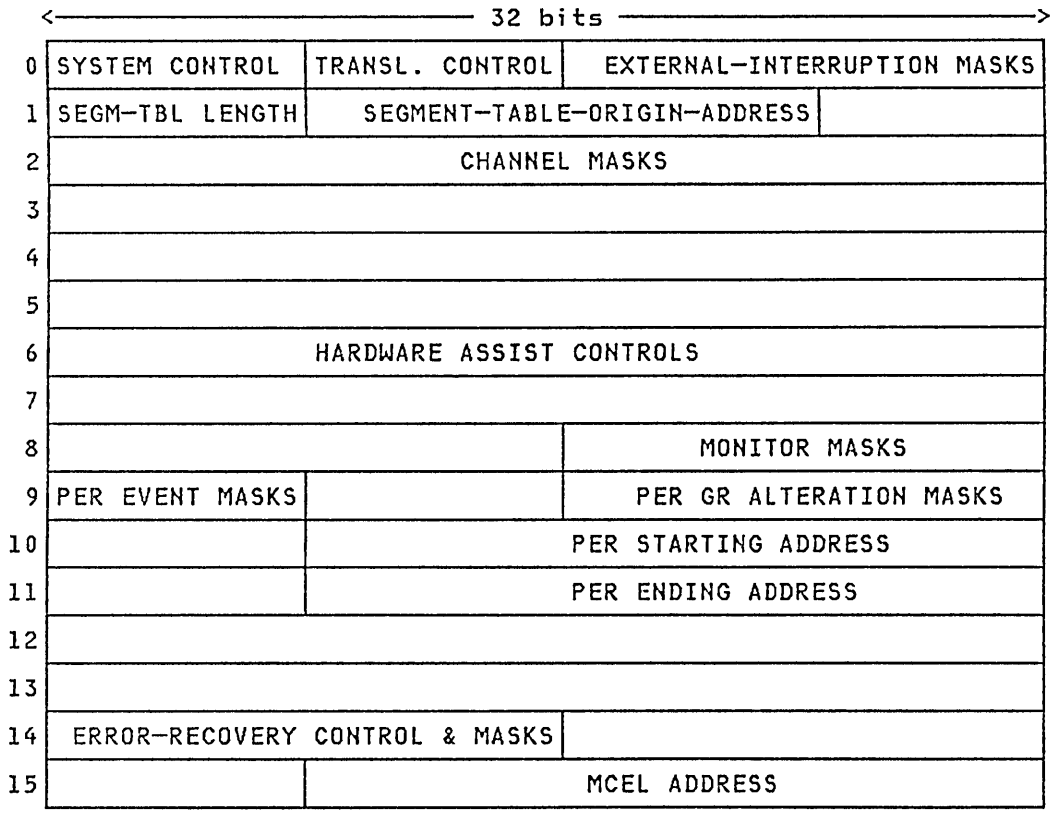



**Figure 72. Control Register Allocation**

| Word | Bits | Name of Field | Associated with | Initial Value |
|---|---|---|---|---|
| 0 | 0 | Block-Multiplex Mode | Block-Multiplex Control | 1 |
| 0 | 1 | SSM Suppression | Extended Control | 0 |
| 0 | 2 | TOD Clock Synchronous Ctrl. | Attached Processing | 0 |
| 0 | 8-9 | Page Size[1] | Dynamic Addr. Translation | 10 |
| 0 | 10 | Reserved | Dynamic Addr. Translation | 0 |
| 0 | 11-12 | Segment size[1] | Dynamic Addr. Translation | 00 |
| 0 | 16 | Malfunction Alter Mask | Attached Processing | 1 |
| 0 | 17 | Emergency Signal Mask | Attached Processing | 1 |
| 0 | 18 | External Call Mask | Attached Processing | 1 |
| 0 | 19 | TOD Synchronous Check Mask | Attached Processing | 1 |
| 0 | 20 | Clock Comparator Mask | Clock Comparator | 1 |
| 0 | 21 | Processor Timer Mask | Processor Timer | 0 |
| 0 | 22 | MSSF Mask | External Interruption | 1 |
| 0 | 24 | Interval Timer Mask | External Interruption | 1 |
| 0 | 25 | Interrupt Key Mask | External Interruption | 1 |
| 0 | 26 | External Signal Mask | External Interruption | 0 |
| 0 | 30 | IUCV | External Interruption | 0 |
| 0 | 31 | VMCF | External Interruption | 0 |
| | | [1]The initial value varies depending upon whether virtual storage is supported in the virtual machine. | | |
| 1 | 0-7 | Segment Table Length | Dynamic Addr. Translation | Set by CP. Value varies with the type of virtual machine. |
| 1 | 8-25 | Segment Table Address | Dynamic Addr. Translation | |
| 2 | 0-31 | Channel Masks | I/O Interruptions | FFFFFFFF. Set to zero on the attached processor in attached processor systems. |

**Figure 73 (Part 1 of 3). Control Register Assignments**

| Word | Bits | Name of Field | Associated with | Initial Value |
|---|---|---|---|---|
| 6 | 0 | VM Assist | Hardware Assist | Value depends upon virtual machine |
| 6 | 1 | VM Problem State | Hardware Assist | Value depends upon virtual machine |
| 6 | 2 | ISK & SSK | Hardware Assist | Value depends upon virtual machine |
| 6 | 3 | S/360 or S/370 instructions | Hardware Assist | Value depends upon virtual machine |
| 6 | 4 | Virtual SVC Interrupts | Hardware Assist | Value depends upon virtual machine |
| 6 | 5 | Shadow Table Fixup | Hardware Assist | Value depends upon virtual machine |
| 6 | 6 | CP Assist | Hardware Assist | Value depends upon virtual machine |
| 6 | 7 | Virtual Interval Timer | Hardware Assist | Value depends upon virtual machine |
| 6 | 8-28 | Real address of VM pointer list | Hardware Assist | Value depends upon virtual machine |
| 8 | 16-31 | Monitor Masks | Monitoring | Value depends upon virtual machine |
| 9 | 0-3 | PER[1] Event Masks | Program-Event Recording | Value depends upon virtual machine. |
| 9 | 16-31 | PER GPR Alteration Masks | Program-Event Recording | |
| | | [1]PER means program-event recording. | | |
| 10 | 8-31 | PER Starting Address | Program-Event Recording | Value depends upon virtual machine. |
| 11 | 8-31 | PER Ending Address | Program-Event Recording | Value depends upon virtual machine. |

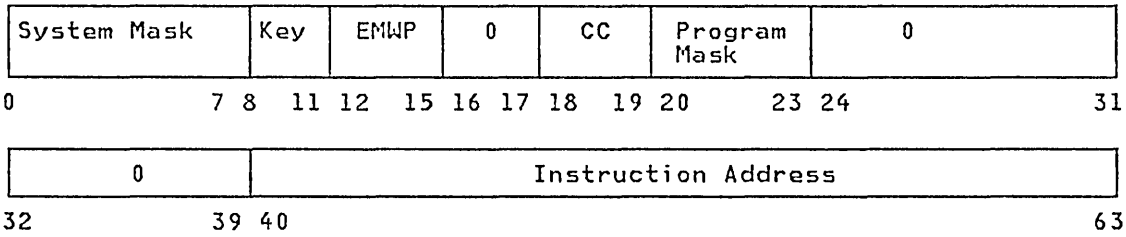
**Figure 73 (Part 2 of 3). Control Register Assignments**

| Word | Bits | Name of Field | Associated with | Initial Value |
|---|---|---|---|---|
| 14 | 0 | Check-Stop Control | Machine-Check Handling | Value depends upon machine check handler for the virtual machine. |
| 14 | 1 | Synchronous MCEL[2] Control | Machine-Check Handling | |
| 14 | 2 | I/O Extended Logout Control | Channel-Check Handling | |
| 14 | 4 | Recovery Report Mask | Machine-Check Handling | |
| 14 | 5 | Degradation Report Mask | Machine-Check Handling | |
| 14 | 6 | External Damage Report Mask | Machine-Check Handling | |
| 14 | 7 | Warning Mask | Machine-Check Handling | |
| 14 | 8 | Asynchronous MCEL Control | Machine-Check Handling | |
| 14 | 9 | Asynchronous Fixed Log Ctrl. [2]MCEL means machine-check extended logout. | Machine-Check Handling | |
| 15 | 8-28 | MCEL Address | Machine-Check Handling | Points to extend I/O logout area |

**Figure 73 (Part 3 of 3). Control Register Assignments**

*Explanation:*

The fields not listed are unassigned.
The initial value of unassigned register positions is unpredictable.




The fields of the 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 74. The Extended Control PSW (Program Status Word)**

# Appendix B. VM/SP Monitor Tape Format and Content

Each time a monitor call interrupt occurs, VM/SP Monitor receives control and collects data appropriate for the particular class and code of MONITOR CALL. (Or, for USER, PERFORM, or DASTAP classes, VM/SP Monitor gets control at periodic intervals to collect data.) The data is formatted into records that are collected sequentially in the order that each interrupt occurred. The tape data format is standard Variable Blocked (VB) format. Data is written at the default tape drive density. Maximum block and record lengths are 4096 bytes. The formats and contents of all the kinds of data records for the currently implemented classes and codes of MONITOR CALL are listed below.

All values described in the following records are binary unless otherwise noted.

[1]indicates that the field is EBCDIC.

[2]Indicates that the field is in special timer format described below.

[3]See *VM/SP Data Areas and Control Block Logic, Volume 1* for field format definition.

## Header Record

Every data record is preceded by the following 12-byte header:

| Data Item | Number of Bytes | DSECT Variable Name |
|---|---|---|
| Total bytes in record | 2 | MNHRECS7 |
| Zeros (standard V format record) | 2 | |
| MONITOR CALL class number | 1 | MNHCLASS |
| MONITOR CALL code number | 2 | MNHCODE |
| Time of Day | 5 | MNHTOD |

**Note:** Time of day occupies 2 fullwords in storage, with the rightmost 12 bits zeros. The rightmost 2 bytes and the leftmost byte are ignored, giving 16-microsecond accuracy instead of 1-microsecond.

The first 4 bytes of this header are the standard variable-format record field.

## Data Records

### Class Zero - Codes for Tape Header, Trailer, and Data Suspension Records

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 97 | Tape header record | | | |
| | CPU serial/model number | 8 | CPUID | MN097CPU |
| | Software version number[1] | 8 | DMKCPEID | MN097LEV |
| | Date of data collection session[1] | 8 | TOD clock | MN097DAT |
| | Time of data collection session[1] | 8 | TOD clock | MN097TIM |
| | Userid of monitor controller[1] | 8 | VMUSER | MN097UID |
| | CR8 mask of enabled classes | 4 | DMKPRGC8 | MN097CR8 |
| | Size of CP nucleus | 4 | Derived by CP | MN097NUC |
| | Size of Free/Fret pools | 4 | Derived by CP | MN097FSS |
| | Size of dynamic paging area | 4 | Derived by CP | MN097DPA |
| | Size of trace table | 4 | Derived by CP | MN097TTS |
| | Size of V=R area (if any) | 4 | Derived by CP | MN097VR |
| | CPU logical address | 2 | LPUADDR | MN097CPL |
| | APU logical address | 2 | LPAUDDRX | MN097APL |
| | Generated system mode | 2 | DMKSYSAP | MN097MOD |
| | Unused | 2 | | |
| | PP map | 8 | DMKCPEPP | MN097CPP |
| 98 | Tape trailer record | | | |
| | Userid of user shutting down monitor[1] | 8 | VMUSER | MN098UID |
| 99 | Tape write suspension record | | | |
| | TOD at suspension[2] | 5 | --- | MN099TOD |
| | Count of write suspensions | 4 | --- | MN099CNT |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00 | Interval statistics | | | |
| | Total main processor idle time[3] | 8 | IDLEWAIT | MN000WID |
| | Total main processor page wait[3] | 8 | PAGEWAIT | MN000WPG |
| | Total main processor time I/O wait[3] | 8 | IONTWAIT | MN000WIO |
| | Total main processor problem time[3] | 8 | PROBTIME | MN000PRB |
| | Total paging start I/Os | 4 | DMKPAGPS | MN000PSI |
| | Total page I/O requests | 4 | DMKPAGCC | MN000CPA |
| | Current page frames on free list | 4 | DMKPTRFN | MN000NFL |
| | Pages being written, due for free list | 4 | DMKPTRSW | MN000PSN |
| | Total pages flushed, but reclaimed | 4 | DMKPTRPR | MN000PRC |
| | Number of reserved pages | 4 | DMKPTRRC | MN000RPC |
| | Number of shared system pages | 4 | DMKPTRSC | MN000SPC |
| | Total number of times free list empty | 4 | DMKPTRF0 | MN000FLF |
| | Total number of calls to DMKPTRFR | 4 | DMKPTRFC | MN000CPT |
| | Total pages stolen from in-queue users | 4 | DMKPTRSS | MN000SS |
| | Number of pages swapped from the flush list | 4 | DMKPTRFF | MN000PFF |
| | Number of pages examined in stealing pages | 4 | DMKPTRRF | MN000PRF |
| | Number of full scans done in stealing pages | 4 | DMKPTRCS | MN000PCS |
| | Total real external interrupts to main processor | 4 | DMKPSANX | MN000NXR |
| | Total calls to DMKPRVLG | 4 | DMKPRVNC | MN000CPR |
| | Total calls to DMKVIOEX | 4 | DMKVSICT | MN000CVI |
| | Total calls to CCWTRANS from DMKVIO | 4 | DMKVSICW | MN000CCW |
| | Total virtual interval timer interrupts reflected | 4 | DMKDSPIT | MN000ITI |
| | Total virtual CPU timer interrupts reflected | 4 | DMKDSPPT | MN000PTI |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00 | Total virtual clock comparator interrupts reflected | 4 | DMKDSPCK | MN000CKI |
| | Total virtual SVC interrupts simulated by main processor | 4 | PSASVCCT | MN000CSV |
| | Total virtual program interrupts handled | 4 | DMKPRGCT | MN000CPG |
| | Total I/O interrupts handled | 4 | DMKIOSCT | MN000CIO |
| | Total calls to dispatch (main) | 4 | DMKDSPCC | MN000CDS |
| | Total fast reflects in dispatch | 4 | DMKDSPAC | MN000CDA |
| | Total dispatches for new PSW | 4 | DMKDSPBC | MN000CDB |
| | Total calls to schedule | 4 | DMKSCHCT | MN000CSC |
| | Count of virtual machine SSK simulated | 4 | DMKPRVEK | MN000EK |
| | Count of virtual machine ISK simulated | 4 | DMKPRVIK | MN000IK |
| | Count of virtual machine SSM simulated | 4 | DMKPRVMS | MN000MS |
| | Count of virtual machine LPSW simulated | 4 | DMKPRVLP | MN000LP |
| | Count of virtual machine diagnose instructions | 4 | DMKPRVDI | MN000DI |
| | Count of virtual machine SIO simulated | 4 | DMKVSISI | MN000SI |
| | Count of virtual machine SIOF simulated | 4 | DMKVSISF | MN000SF |
| | Count of virtual machine TIO simulated | 4 | DMKVSITI | MN000TI |
| | Count of virtual machine CLRIO simulated | 4 | DMKVSICI | MN000CI |
| | Count of virtual machine HIO simulated | 4 | DMKVSIHI | MN000HI |
| | Count of virtual machine HDV simulated | 4 | DMKVSIHD | MN000HD |
| | Count of virtual machine TCH simulated | 4 | DMKVSITC | MN000TC |
| | Count of virtual machine STNSM simulated | 4 | DMKPRVMN | MN000MN |
| | Count of virtual machine STOSM simulated | 4 | DMKPRVMO | MN000MO |
| | Count of virtual machine LRA simulated | 4 | DMKPRVLR | MN000LR |
| | Count of virtual machine STIDP simulated | 4 | DMKPRVCP | MN000CP |
| | Count of virtual machine STIDC simulated | 4 | DMKPRVCH | MN000CH |
| | Count of virtual machine SCK simulated | 4 | DMKPRVTE | MN000TE |
| | Count of virtual machine SCKC simulated | 4 | DMKPRVCE | MN000CE |
| | Count of virtual machine STCKC simulated | 4 | DMKPRVCT | MN000CT |
| | Count of virtual machine SPT simulated | 4 | DMKPRVPE | MN000PE |
| | Count of virtual machine STPT simulated | 4 | DMKPRVPT | MN000PT |
| | Count of virtual machine SPKA simulated | 4 | DMKPRVEP | MN000EP |
| | Count of virtual machine IPK simulated | 4 | DMKPRVIP | MN000IP |
| | Count of virtual machine PTLB simulated | 4 | DMKPRVPB | MN000PB |
| | Count of virtual machine RRB simulated | 4 | DMKPRVRR | MN000RR |
| | Count of virtual machine STCTL simulated | 4 | DMKPRVTC | MN000TCL |
| | Count of virtual machine LCTL simulated | 4 | DMKPRVLC | MN000LCL |
| | Count of virtual machine CS simulated | 4 | DMKPRVCS | MN000CS |
| | Count of virtual machine CDS simulated | 4 | DMKPRVCD | MN000CD |
| | Count of virtual machine diagnose disk I/O | 4 | DMKHVCDI | MN000HDI |
| | Number of users dialed to virtual machines | 4 | DMKSYSND | MN000NDU |
| | Number of users logged on | 4 | DMKSYSNM | MN000NAU |
| | Number of page reads by main processor | 4 | PGREAD | MN000PRD |
| | Number of page writes by main processor | 4 | PGWRITE | MN000PWR |
| | Number of system pageable pages | 4 | DMKDSPNP | MN000NPP |
| | Sum of working sets of in-queue users | 4 | DMKSCNPU | MN000SWS |
| | Number of users in interactive queue (Q1) | 4 | DMKSCHN1 | MN000Q1N |
| | No. of users in compute-bound queue (Q2) | 4 | DMKSCHN2 | MN000Q2N |
| | Number of users eligible to enter Q1 | 2 | DMKSCHW1 | MN000Q1E |
| | Number of users eligible to enter Q2 | 2 | DMKSCHW2 | MN000Q2E |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| | Monitor sampling interval (seconds) | 2 | DMKPRGTI | MN000INT |
| | Count of cylinders allocated on primary paging device | 2 | ALOCUSED | MN000PPA |
| | Cylinder capacity of primary paging device | 2 | ALOCMAX | MN000PPC |
| | Reserved | 2 | --- | MN0RSV1 |
| | Count of mini IOB stack depletes | 2 | DMKIOSNM | MN000ISD |
| | Count of mini IOB enqueues | 4 | | MN000GTM |
| | Count of mini IOB dequeues | 4 | | MN000DQM |
| | Count of SIOs on alternate paths | 4 | | MN000SWP |
| | Count of FREE/FRET extends | 4 | DMKFRENP | MN000EXT |
| | Count of FREE/FRET unextends | 4 | | MN000NXT |
| | Count of attempts to split subpool | 4 | | MN000ATT |
| | Count of SUBPOOL SPLITS | 4 | | MN000CNT |
| | | 4 | | |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 01 | Internal statistics for attached processor | | | |
| | Total attached processor idle wait time | 8 | IDLEWAIT | MN001WID |
| | Total attached processor page wait time | 8 | PAGEWAIT | MN001WPG |
| | Total attached processor I/O wait time | 8 | IONTWAIT | MN001WIO |
| | Total attached processor problem time | 8 | PROBTIME | MN001PRB |
| | Total real external interrupts for attached processor | 4 | DMKPSANX | MN001NXR |
| | Total SVCs reflected by attached processor | 4 | PSASVCCT | MN001CSV |
| | Page reads by attached processor | 4 | PGREAD | MN001PRD |
| | Page writes by attached processor | 4 | PGWRITE | MN001PWR |
| | Total time spin on system lock | 4 | DMKLOKSY+8 | MN001SSY |
| | Number of spins on system lock | 4 | DMKLOKSY+12 | MN001NSY |
| | Total time spin on DMKFRE lock | 4 | DMKLOKFR+8 | MN001SFR |
| | Number of spins on DMKFRE lock | 4 | DMKLOKFR+12 | MN001NFR |
| | Total time spin on RUNLIST lock | 4 | DMKLOKRL+8 | MN001SRN |
| | Number of spins on RUNLIST lock | 4 | DMKLOKRL+12 | MN001NRN |
| | Total time spin on timer request lock | 4 | DMKLOKTR+8 | MN001STM |
| | Number of spins on timer request lock | 4 | DMKLOKTR+12 | MN001NTM |
| | Total time spin on dispatcher queue lock | 4 | DMKLOKDS+8 | MN001SDP |
| | Number of spins on dispatcher queue lock | 4 | DMKLOKDS+12 | MN001NDP |
| | Number of times CPFRELK set | 4 | | MN001NFL |
| | Number of times CPFRESW set | 4 | | MN001NFS |
| | Number of times system lock deferred | 4 | LOKSYSCT | MN001NSD |
| | Number of times VMBLOK lock deferred | 4 | LOKVMCT | MN001NVD |
| | Number of DMKDSPRU entries | 4 | | MN001NRU |
| | Total time spin on I/O lock | 4 | DMKLOKIO | MN001SIO |
| | Total no. spins for I/O lock | 4 | DMKLOKIO | MN001NIO |
| | Total time spin on RM lock | 4 | DMKLOKRM | MN001SRM |
| | Total no. spins for RM lock | 4 | DMKLOKRM | MN001NRM |
| | No. quiesce ems on IPL proc | 4 | DMKEMSCT | MN001NQ1 |
| | No. quiesce ems on non-IPL proc | 4 | DMKEMSCT | MN001NQ2 |
| | No. extend ems on IPL proc | 4 | DMKEMSCT | MN001NE1 |
| | No. extend ems on non-IPL proc | 4 | DMKEMSCT | MN001NE2 |
| | No. resume XC on IPL proc | 4 | DMKXCCTS | MN001NR1 |
| | No. resume XC on non-IPL proc | 4 | DMKXCCTS | MN001NR2 |
| | No. dispatch XC on IPL proc | 4 | DMKXCCTS | MN001ND1 |
| | No. dispatch XC on non-IPL proc | 4 | DMKXCCTS | MN001N02 |
| | No. dispatch XC on non-IPL proc | 4 | DMKXCCTS | MN001ND2 |
| | No. wakeup XC on IPL proc | 4 | DMKXCCTS | MN001NW1 |
| | No. wakeup XC on non-IPL proc | 4 | DMKXCCTS | MN001NW2 |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 02 | Average queue delay | 4 | DMKSCHQT | MN002SQT |
| | Average eligible list time | 4 | DMKSCHET | MN002SET |
| | Average utilization | 4 | DMKSCHFS | MN002SFS |
| | Average resident page request | 4 | DMKSCHAP | MN002SAP |
| | Average desired processor/page read | 4 | DMKSCHKA | MN002SKA |
| | Average processor overhead/page read | 4 | DMKSCHUC | MN002SUC |
| | Calculated paging bias | 4 | DMKSCHPB | MN002SPB |
| | Paging bias limit | 4 | | |
| | Interactive bias | 4 | DMKSCHIB | MN002SIB |
| | Count of Q3 users | 4 | DMKSCHQ3 | MN002SQ3 |
| | Q1 in-queue count | 8 | VMQTOD | MN002Q11 |
| | Q1 in-queue time | 8 | VMQELP | MN002Q12 |
| | Q1 eligible list time | 8 | VMQWT | MN002Q13 |
| | Q1 in-queue processor time | 8 | VMQCPU | MN002Q14 |
| | Q1 estimated average pages per second | 8 | VMQPGS | MN002Q15 |
| | Q1 count of queue drops | 4 | VMQCNT | MN002Q16 |
| | Q1 in-queue page reads | 4 | VMQPRD | MN002Q17 |
| | Q1 in-queue page steals | 4 | VMQSTL | MN002Q18 |
| | Reserved | 4 | --- | MN00RSV1 |
| | Q2 in-queue time stamp | 8 | VMQTOD | MN002Q21 |
| | Q2 in-queue time | 8 | VMQELP | MN002Q22 |
| | Q2 eligible list time | 8 | VMQWT | MN002Q23 |
| | Q2 in-queue processor time | 8 | VMQCPU | MN002Q24 |
| | Q2 estimated average pages per second | 8 | VMQPGS | MN002Q25 |
| | Q2 count of queue drops | 4 | VMQCNT | MN002Q26 |
| | Q2 in-queue page reads | 4 | VMQPRD | MN002Q27 |
| | Q2 in-queue page steals | 4 | VMQSTL | MN002Q28 |
| | Reserved | 4 | --- | MN00RSV2 |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 03 | No. of calls to migrate | 4 | DMKSCHQ1 | MN003CMG |
| | Times migration limit halved | 4 | DMKSCHQ1 | MN003TLH |
| | Times limit was quartered | 4 | DMKSCHN1 | MN003TLQ |
| | Times a user was selected | 4 | DMKSCHW1 | MN003TUS |
| | No. migrations by command | 4 | | MN003MBC |
| | No. calls resulting in migration | 4 | | MN003CRM |
| | No. users moved | 4 | | MN003NUM |
| | No. segments moved | 4 | | MN003NSM |
| | No. pages moved | 4 | | MN003NPM |
| | No. full disks moved | 4 | | MN003NDM |
| | Calls to restore swaptable | 4 | | MN003CSR |
| | Calls to migrate swaptable | 4 | | MN003CSM |
| | No. of tables migrated | 4 | | MN003NTM |
| | No. of tables restored | 4 | | MN003NTR |
| | Calls to pseudo translator | 4 | | MN003CPT |
| | Reserved | 4 | | MN003RSV |
| | Total test protect ins simulated | 4 | DMKPRVTP | MN003CTP |
| | Total IPTE instructions simulated | 4 | DMKPIPTE | MN003CIP |
| | No. preferred FH pages available | 4 | DMKPGTDM | MN003CDM |
| | No. preferred MH pages available | 4 | DMKPGTDK | MN003CDK |
| | No. preferred MH pages allocated | 4 | DMKPGTPC | MN003CPC |
| | Limit of preferred MH pages | 4 | DMKPGTPL | MN003CPL |
| | % value for SET SRM MHFULL | 4 | DMKPGTPN | MN003CPN |
| | Unused | 4 | | MN003CUN |

**Note:** Privileged instructions simulated by the fast path simulation routines (DMKFSP) are not recorded.

**Class One - RESPONSE**

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00 | Read command sent to terminal | | | |
| | Userid | 8 | VMUSER | MN10XUID |
| | Line address | 2 | | MN10XADD |
| 01 | Terminal output line | | | |
| | userid | 8 | VMUSER | MN10XUID |
| | Line address | 2 | | MN10YADD |
| | Byte count | 1 | | MN10YCNT |
| | Line of data | Variable | | MN10YIO |
| 02 | Edited terminal input line | | | |
| | Userid | 8 | VMUSER | MN10XUID |
| | Line address | 2 | | MN10XADD |
| | Byte count | 1 | | MN10YCNT |
| | Line of data[1] | Variable | | MN10YIO |
| 03 | Sleep issued with time out | | | |
| | Userid | 8 | VMUSER | MN10XUID |
| | Line address | 2 | | MN10XADD |
| 04 | Terminal logged on | | | |
| | Userid | 8 | VMUSER | MN10XUID |
| | Line address | 2 | | MN10XADD |
| 05 | Terminal logged off | | | |
| | Userid | 8 | VMUSER | MN10XUID |
| | Line address | 2 | | MN10XADD |

Note that the line addresses for the 370X in NCP mode appear as the base address.

These records are created at the time that DMKQCN handles the console I/O request. This may reflect a slightly different time than that of the SIO or the I/O interrupt. If DMKQCN is called to write a line that is longer than Terminal line size, more than one MC is issued, resulting in more than one record. Input and output terminal data collected is limited to 128 bytes. Longer lines are truncated.

**Class Two - SCHEDULE**

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 02 | User dropped from dispatch queue | | | |
| | Userid[1] | 8 | VMUSER | MN20XUID |
| | Number of system pageable pages | 4 | DMKDSPNP | MN20XNPP |
| | Sum of working sets of in-queue users | 4 | DMKSCHPU | MN20XSWS |
| | No. of users in interactive queue (Q1) | 4 | DMKSCHN1 | MN20XQ1N |
| | No. of users in compute-bound queue (Q2) | 4 | DMKSCHN2 | MN20XQ2N |
| | Number of users eligible for Q1 | 2 | DMKSCHW1 | MN20XQ1E |
| | Number of users eligible for Q2 | 2 | DMKSCHW2 | MN20XQ2E |
| | User new projected working set size | 2 | VMWSPROJ | MN20XWSS |
| | Queue being dropped from (1 or 2) | 1 | Q1DROP | MN20XQNM |
| | Processor address | 1 | --- | MN20XPRC |
| | Accumulated user CP simulation time[3] | 8 | VMTTIME | MN20YTTI |
| | Accumulated user virtual time[3] | 8 | VMVTIME | MN20YVTI |
| | Eligible list priority | 4 | VMQPRIOR | MN204PRI |
| | Pages read while in queue | 2 | VMPGREAD | MN202PGR |
| | Sum of pages resident at all reads | 2 | VMPGRINQ | MN202APR |
| | No. of pages referenced while in queue | 2 | --- | MN202REF |
| | Current number of pages resident | 2 | VMPAGES | MN202RES |
| | Number of pages stolen while in queue | 2 | VMSTEALS | MN202PST |
| | User total virt non-spool device SIO count | 4 | VMIOCNT | MN202IOC |
| | User total virtual cards punched | 4 | VMPNCH | MN202PNC |
| | User total virtual lines printed | 4 | VMLINS | MN202LIN |
| | User total virtual cards read | 4 | VMCRDS | MN202CRD |
| | User last executed on this processor | 1 | VMLSTPRC | MN202LPR |
| 03 | User added to dispatch queue | | | |
| | Userid | 8 | VMUSER | MN20XUID |
| | Number of system pageable pages | 4 | DMKDSPNP | MN20XNPP |
| | Sum of working sets of in-queue users | 4 | DMKSCHPU | MN20XSWS |
| | Number of users in interactive queue (Q1) | 4 | DMKSCHN1 | MN20XQ1N |
| | No. of users in compute-bound queue (Q2) | 4 | DMKSCHN2 | MN20XQ2N |
| | Number of users eligible for Q1 | 2 | DMKSCHW1 | MN20XQ1E |
| | Number of users eligible for Q2 | 2 | DMKSCHW2 | MN20XQ2E |
| | User's projected working set size | 2 | VMWSPROJ | MN20XWSS |
| | Queue being added to | 1 | gen reg 15 | MN20XQNM |
| | Processor address (main or attached) | 1 | --- | MN20XPRC |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 04 | User added to eligible list | | | |
| | Userid | 8 | VMUSER | MN20XUID |
| | Number of system pageable pages | 4 | DMKDSPNP | MN20XNPP |
| | Sum of working sets of in-queue users | 4 | DMKSCHPU | MN20XSWS |
| | Number of users in interactive queue (Q1) | 4 | DMKSCHN1 | MN20XQ1N |
| | No. of users in compute-bound queue (Q2) | 4 | DMKSCHN2 | MN20XQ2N |
| | Number of users eligible for Q1 | 2 | DMKSCHW1 | MN20XQ1E |
| | Number of users eligible for Q2 | 2 | DMKSCHW2 | MN20XQ3E |
| | User's projected working set size | 2 | VMWSPROJ | MN20XWSS |
| | Queue being added to | 1 | VMQ1 | MN20XQNM |
| | Processor address (main or attached) | 1 | --- | MN20XPRC |
| | Accumulated user CP simulation time | 8 | VMTTIME | MN20YTTI |
| | Accumulated user virtual time | 8 | VMVTIME | MN20YVTI |
| | Eligible list priority | 2 | VMEPRIOR | MN20YPRI |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00 | Interval user resource utilization statistics | | | |
| | Userid[1] | 8 | VMUSER | MN400UID |
| | Accumulated user CP simulation time | 8 | VMTTIME | MN400TTI |
| | Accumulated user virtual time | 8 | VMVTIME | MN400VTI |
| | Total page reads | 4 | VMPGREAD | MN400PGR |
| | Total page writes | 4 | VMPGWRIT | MN400PGW |
| | Total non-spooled I/O requests | 4 | VMIOCNT | MN400IOC |
| | Total cards punched | 4 | VMPNCH | MN400PNC |
| | Total lines printed | 4 | VMLINS | MN400LIN |
| | Total cards read | 4 | VMCRDS | MN400CRD |
| | User running status | 1 | VMRSTAT | MN400RST |
| | User dispatch status | 1 | VMDSTAT | MN400DST |
| | User operating status | 1 | VMOSTAT | MN400OST |
| | User queuing status | 1 | VMQSTAT | MN400QST |
| | User processing status | 1 | VMPSTAT | MN400PST |
| | User control status | 1 | VMESTAT | MN400EST |
| | User tracing control | 1 | VMTRCTL | MN400TST |
| | User message level | 1 | VMMLEVEL | MN400MLV |
| | User queue level | 1 | VMQLEVEL | MN400QLV |
| | User command level | 1 | VMCLEVEL | MN400CLV |
| | User timer level | 1 | VMTLEVEL | MN400TLV |
| | Interrupt pending summary | 1 | VMPEND | MN400PND |
| | User's externally assigned priority | 1 | VMUPRIOR | MN400UPR |
| | Reserved | 1 | --- | MN4RSV1 |
| 00 | Current number of pages resident | 2 | VMPAGES | MN400RES |
| | Current working set size estimate | 2 | VMWSPROJ | MN400WSS |
| | Page frames allocated on drum | 2 | VMPDRUM | MN400PDR |
| | Page frames allocated on disk | 2 | VMPDISK | MN400PDK |
| | Monitor sampling interval (seconds) | 2 | DMKPRGTI | MN400INT |
| | User last executed on this processor | 1 | VMLSTPRC | MN400LPR |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00 | Start of PRIVOP simulation | | | |
| | Userid[1] | 8 | VMUSER | MN500UID |
| | The privileged instruction | 4 | VMINST | MN500INS |
| | Virtual storage address of PRIVOP | 4 | VMPSW | MN500VAD |
| | Total user CP simulation time at start of simulation | 8 | CPU timer | MN500OVH |

**Note:** Privileged instructions simulated by the fast path simulation routines (DMKFSP) are not recorded.

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00,01 | Device activity data for all Tape and DASD devices | | | |
| | Number of device blocks recorded | 2 | | MN600NUM |
| | For each device - | | | |
| | Device address | | RDEVADDR<br>RCUADDR | |
| | | 2 | RCHADDR | MN600ADD |
| | Type codes | 2 | RDEVTYPC | MN600TY |
| | Volume serial number[1] | 6 | RDEVSER | MN600SER |
| | Device accumulated I/O count | 4 | RDEVIOCT | MN600CNT |

Note:

The monitor code 0 record is collected when the MONITOR START TAPE command is entered. Thereafter, all DASTAP records are collected with a monitor code of 1.

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 02 | No. samples for interval IPL proc | 2 | MNCHSAM1 | MN602SAM |
| | No. samples for interval non-IPL proc | 2 | MNCHSAM2 | MN602SA2 |
| | Device address | 2 | RDEVADD | MN602ADD |
| | No. times control unit busy | 2 | MNCUBSY | MN602CUB |
| | No. times device busy IPL proc | 2 | MNDVBSY | MN602DVB |
| | I/O tasks queued on control unit | 2 | RCUQCNT | MN602CUQ |
| | I/O tasks queued on device | 1 | RDEVQCNT | MN602DVQ |
| | No. times device busy non-IPL proc | 1 | MDVBSY2 | MN602DV2 |
| 03 | Channel busy counts IPL proc | 32 | MCHDAT1 | MN603CBI |
| | I/O tasks queued on channel IPL proc | 32 | RCHQCNT | MN603CQI |
| | Channel busy counts non-IPL proc | 32 | MCHDAT2 | MN603CB2 |
| | I/O tasks queued on channel non-IPL proc | 32 | RCHQCNT | MN603CQ2 |

**Class Seven - SEEKS**

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00 | DASD I/O request record | | | |
| | Userid[1] | 8 | VMUSER | MN700UID |
| | Device address | | RDEVADDR<br>RCUADDR | |
| | | 2 | RCHADDR | MN700ADD |
| | Seek cylinder address | 2 | IOBCYL | MN700CYL |
| | Current arm position | 2 | RDEVCYL | MN700CCY |
| | Number of queued I/O tasks on device | 1 | RDEVQCNT | MN700QDV |
| | Number of queued I/O tasks on control unit | 1 | RCUQCNT | MN700QCU |
| | Number of queued I/O tasks on channel | 1 | RCHQCNT | MN700QCH |
| | Current seek direction | 1 | RDEVFLAG | MN700DIR |
| | Processor address | 2 | RCHPROC | MN700PRO |

**Note:**

Current seek direction value is

X'00' seeking to lower cylinder address
X'01' seeking to higher cylinder address

**Class Eight - SYSPROF -- Additional data for system profile class**

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 02 | Additional data at add queue, drop queue times | | | |
| | Number of 4-byte device block counts which follow | 2 | --- | MN802NUM |
| | For each device ...count of I/O's | 4 | RDEVIOCT | --- |
| | After device counts ... | | | |
| | Current number of users logged on | 4 | DMKSYSNM | MN802NAU |
| | Total system page reads | 4 | PGREAD | MN802PGR |
| | Total system page writes | 4 | PGWRITE | MN802PGW |
| | Current number of pageable pages | 4 | DMKDSPNP | MN802NPP |
| | Total system idle time | 8 | IDLEWAIT | MN802WID |
| | Total system page wait time | 8 | PAGEWAIT | MN802WPG |
| | Total system I/O wait time | 8 | IONTWAIT | MN802WIO |
| | Total system problem time | 8 | PROBTIME | MN802PRB |

# Appendix C. CMS Macro Library

The following is a list and brief description of the CMS macros applicable to VM/SP.

Asterisk (*) indicates that the macro is reserved for IBM use.

| CMS Macro | Function |
|---|---|
| *ADT | Generates a CSECT or DSECT for an active disk table. |
| *ADTGEN | Generates an active disk table (ADT) for a disk; used by ADTSECT. |
| *ADTSECT | Generates all the ADTs for CMS. |
| *AFT | Generates a DSECT for an active file table. |
| *AFTSECT | Generates all the AFTs for CMS. |
| BATLIMIT | Table of CPU, punch, and printer limits for user jobs running under CMS batch. |
| BBOX | DSECT of boundary box; contains beginning and ending addresses of background communication region. |
| BGCOM | DSECT of background communication region. |
| BGTCB | Task Control Block. |
| *CMSAVE | Equivalent to SVCSAVE macro. |
| *CMSCB | Generates a list of simulated OS control blocks. |
| *CMSCVT | Generates the communication vector table as supported by CMS. |
| *CMSLEVEL | Defines the value of 'release number' of the feature or program product returned by QUERY CMSLEVEL. Refer to the CMSLEVEL macro for more information. |
| COMPSWT | Sets the compiler switch on or off. Refer to *VM/SP CMS Command and Macro Reference.* |
| *CORG | Sets the origin for CSECT. |
| *DBGSECT | Generates a CSECT or DSECT for DEBUG environment variables. |
| DESTYP | Used by the XEDIT module DMSXIN to determine filetype default settings. The DESTYP block is defined in DMSXTF. |
| *DEVGEN | Generates a device table for a given device; used by the DEVTAB macro. |
| *DEVSECT | DSECT for a device table. |
| *DEVTAB | Generates the device tables for the CMS nucleus. |
| *DIAG | Issues a specified CP Diagnose instruction. |
| DIB | Disk Information Blocks. |
| *DIOSECT | Generates a CSECT or DSECT for all I/O information. |
| DISPW | Generates the calling sequence for the display terminal interface. Refer to the *VM/SP System Programmer's Guide.* |
| DMSABN | ABEND the virtual machine. Refer to the *VM/SP System Programmer's Guide.* |
| *DMSCCB | DSECT describes field of DOS command control block (CCB). Refer to *VM/SP Data Areas and Control Block Logic, Volume 2 (CMS).* |
| *DMSABW | Allocates a work area for DMSABN. |
| *DMSDM | Reserved for IBM use. |

| CMS Macro | Function |
|---|---|
| *DMSERR | Sets up parameter list to type out a CMS error message; Refer to the LINEDIT macro. |
| *DMSERT | DMSERR work area DSECT. |
| DMSEXS | Execute an instruction without nucleus protection. Refer to *VM/SP System Logic and Problem Determination Guide--Volume 2.* |
| DMSFREE | Gets free storage. Refer to the *VM/SP System Programmer's Guide.* |
| *DMSFRES | Calls system free storage service routines. |
| DMSFRET | Releases free storage. Refer to the *VM/SP System Programmer's Guide.* |
| *DMSFREX | Calls system free storage service routines. |
| *DMSFRT | Generates a DSECT for free storage management work area. |
| *DMSFRX | Submacro called by DMSFRET. |
| DMSFST | Sets up a file status table for a given file. Refer to the *VM/SP System Programmer's Guide.* |
| DMSKEY | Sets nucleus protection on or off. Refer to *VM/SP System Logic and Problem Determination Guide--Volume 2.* |
| *DMSLN | Called by DMSERR, LINEDIT macros. |
| *DMSLNC | Called by DMSERR, LINEDIT macros. |
| *DMSLND | Called by DMSERR, LINEDIT macros. |
| *DMSLNP | Called by DMSERR, LINEDIT macros. |
| *DMSLNU | Called by DMSERR, LINEDIT macros. |
| *DMSLNY | Called by DMSERR, LINEDIT macros. |
| *DMSLNZ | Called by DMSERR, LINEDIT macros. |
| *DMSPID | Passes a fileid in quotes into separate filename, filetype, filemode, used by FSCB, and FSPOINT. |
| *DMSTMS | Used by RDTAPE, WRTAPE, and TAPECTL. |
| DOSAVE | DSECT, describes fields in the logical transient area (LTA). |
| DOSCB | DOS simulation control block used for simulation of the CMS file control block (FCB). |
| DOSCON | Creates CMS/DOS control blocks for DMSNUC. |
| DTFSD | DTFSD DSECT. |
| DTFX | DTF extension DSECT. |
| *EDCB | Frees storage control blocks initialized by DMSEDX for CMS edit modules. |
| *EPLIST | DSECT to map extended plist passed in register 0. |
| *EQUATES | Generates CMS equates for symbolic names. |
| *EXCP | Issues an SVC 0. |
| *EXTSECT | Defines storage for the timer interrupt. |
| *FCB | Generates a file control block (FCB) DSECT. |
| FSCB | Sets up a file system control block. Refer to the *VM/SP CMS Command and Macro Reference.* |
| *FSCBD | DSECT that describes fields in CMS PLIST for related commands. |

| CMS Macro | Function |
|---|---|
| FSCLOSE | Closes a file. Refer to the *VM/SP CMS Command and Macro Reference.* |
| *FSENTR | Used by CMS file system routines at entry. |
| FSERASE | Erases a file. Refer to the *VM/SP CMS Command and Macro Reference.* |
| FSOPEN | Opens a file. Refer to the *VM/SP CMS Command and Macro Reference.* |
| *FSPOINT | Executes the CMS POINT function. |
| FSREAD | Reads a record from a file. Refer to the *VM/SP CMS Command and Macro Reference.* |
| FSSTATE | Checks for an existing file. Refer to the *VM/SP CMS Command and Macro Reference.* |
| *FSTB | Generates a file status table (file directory) block. |
| *FSTD | Entry to the file status table (file directory) block. |
| FSWRITE | Writes a record into a disk file. Refer to the *VM/SP CMS Command and Macro Reference.* |
| *FVS | Defines storage for file system variables. |
| *GETADT | Gets a specified active disk table. |
| *GETFST | Gets a specified file status table. |
| HNDEXT | Handles external and timer interrupts. Refer to the *VM/SP CMS Command and Macro Reference.* |
| HNDINT | Handles interrupt on devices. Refer to the *VM/SP CMS Command and Macro Reference.* |
| HNDSVC | Handles SVCs. Refer to the *VM/SP CMS Command and Macro Reference.* |
| IJJHCPL | Common VTOC handler input PLIST. |
| IJJHDLST | Common VTOC handler descriptor list DSECT. |
| IJJHMFT1 | Format 1 VTOC label DSECT. |
| *IO | Contains PLISTs needed to access CMS I/O routines. |
| *IOSECT | Defines miscellaneous I/O variables. |
| *KEYSECT | Contains variables necessary for storage key handling. |
| *KXCHK | Checks to see if HX has been entered by the user. |
| LABREC | DLBL/EXTENT record. |
| *LDM | Loads double multiple (for floating point registers). |
| *LDRST | CMS Loader work area. |
| LINEDIT | Types a line to the terminal. Refer to the *VM/SP CMS Command and Macro Reference.* |
| LOCKTAB | LOCK/UNLOCK resource table. |
| LPLDCT | LABEL macro PLIST. |
| LSCREEN | Used by XEDIT modules to describe the layout of a logical screen on the physical screen. LSCREEN is built by module DMSXSD. |
| *NUCON | Generates a DSECT CMS nucleus constant area. |
| OCTS | OPEN/CLOSE transient SVA PLIST. |
| *OVSECT | DMSOVS work area. |

| CMS Macro | Function |
|---|---|
| *OSFST | Defines an OS file status table for OS ACCESS. |
| *PDSSECT | DSECT used for processing MACLIB files. |
| *PGMSECT | Defines work area for DMSITP. |
| PIBTAB | DSECT, program information block. |
| PIB2TAB | DSECT, program information block extension. |
| PRINTL | Prints a line on the printer. Refer to the *VM/SP CMS Command and Macro Reference.* |
| PRSCB | Used by the XEDIT subcommands PRESERVE and RESTORE. It is built by module DMSXCT. |
| PUNCHC | Punches a card. Refer to the *VM/SP CMS Command and Macro Reference.* |
| RDCARD | Reads a card from the reader. Refer to the *VM/SP CMS Command and Macro Reference.* |
| RDTAPE | Reads a record from tape. Refer to the *VM/SP CMS Command and Macro Reference.* |
| RDTERM | Reads a record from the terminal. Refer to the *VM/SP CMS Command and Macro Reference.* |
| RECSAVE | Used by XEDIT modules to describe the address list for nested macro calls. It is built by DMSXMA. |
| REGEQU | Generates symbolic register equates. Refer to the *VM/SP CMS Command and Macro Reference.* |
| *RELPAGES | Sets the release pages flag. |
| REQDES | Used by XEDIT modules to describe all XEDIT subcommands and their operands and syntax. The REQDES block is defined in DMSXTB. |
| SAVEREG | Used by XEDIT modules to save register contents during subroutine calls. |
| *STDM | Storage for multiple floating-point registers. |
| STRINIT | Initializes storage. Refer to the *VM/SP CMS Command and Macro Reference.* |
| *SUBSECT | CSECT or DSECT for CMS SUBSET use. |
| *SVCENT | Issues a DMSKEY macro before calling an instruction. |
| *SVCSAVE | System save area. |
| *SVCSECT | Defines work area for DMSITS. |
| SYNSUB | Used by XEDIT modules to describe the synonyms defined for XEDIT subcommands. A SYNSUB block is built dynamically by DMSXDC each time a synonym is defined. |
| SYSCOM | DSECT of system communication region. |
| *SYSLOAD | Puts in a specified register the address of a specified routine in NUCON. |
| *SYSNAMES | Saves system names table loaded via CMS routines. |
| TAPECTL | Positions a tape. Refer to the *VM/SP CMS Command and Macro Reference.* |
| *TSOBLKS | Contains CPPL, UPT, PSCB, and the ECT for TSO service routines. |
| *TSOGET | Gets the address of the TSO command processor parameter list (CPPL). |
| *USE | Generates assembler USING and DROP instructions, as needed. |
| *USERSECT | Creates user work area. |

| CMS Macro | Function |
|---|---|
| WAITD | Waits until the next interrupt occurs for the specified device. Refer to the *VM/SP CMS Command and Macro Reference.* |
| WAITT | Waits until all pending I/O to the terminal has completed. Refer to the *VM/SP CMS Command and Macro Reference.* |
| WRTAPE | Writes a record to tape. Refer to the *VM/SP CMS Command and Macro Reference.* |
| WRTERM | Writes a record to the terminal. Refer to the *VM/SP CMS Command and Macro Reference.* |
| ZDESC | Used by XEDIT modules to describe file characteristics. |
| ZFONC | Used by XEDIT modules as a common work area. It is built by DMSXBG only once in an editing session. |
| ZMACST | Used by XEDIT modules to describe an XEDIT macro in storage. A ZMACST block is built dynamically by DMSXMA each time a macro is invoked. |
| ZPACK | Used by XEDIT modules when a file is being packed or unpacked. It is built by DMSXIN or DMSXFD. |

# Index

## Special Characters



## A

## D

## E



## F

## J



## L

## M

## N



## O

## P

## Q



## R

## S

## T

## U



## V

## W



## X



## Z



## 1

## 3

VM/SP System Programmer's Guide (File No. S370/4300-36) Printed in U.S.A. SC19-6203-2

SC19-6203-2

VM/SP
System Programmer's Guide
SC19-6203-2

**READER'S COMMENT FORM**

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

**Note:** *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

| | *Yes* | *No* |
|---|---|---|
| • Does the publication meet your needs? | ☐ | ☐ |
| • Did you find the material: | | |
| Easy to read and understand? | ☐ | ☐ |
| Organized for convenient use? | ☐ | ☐ |
| Complete? | ☐ | ☐ |
| Well illustrated? | ☐ | ☐ |
| Written for your technical level? | ☐ | ☐ |

• What is your occupation? _____

• How do you use this publication:

| | | | |
|---|---|---|---|
| As an introduction to the subject? | ☐ | As an instructor in class? | ☐ |
| For advanced knowledge of the subject? | ☐ | As a student in class? | ☐ |
| To learn about operating procedures? | ☐ | As a reference manual? | ☐ |

**Your comments:**

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

*If you would like a reply, please supply your name and address on the reverse side of this form.*

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

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 PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

Fold | Fold

If you would like a reply, *please print:*

*Your Name* _____

*Company Name* _____ *Department* _____

*Street Address* _____

*City* _____

*State* _____ *Zip Code* _____

*IBM Branch Office serving you* _____

IBM®

VM/SP System Programmer's Guide (File No. S370/4300-36) Printed in U.S.A. SC19-6203-2

SC19-6203-2

VM/SP System Programmer's Guide (File No. S370/4300-36) Printed in U.S.A. SC19-6203-2