IBM

Virtual Machine/
System Product

System Logic and Problem Determination Guide Volume 1 (CP)

Release 5

LY20-0892-4

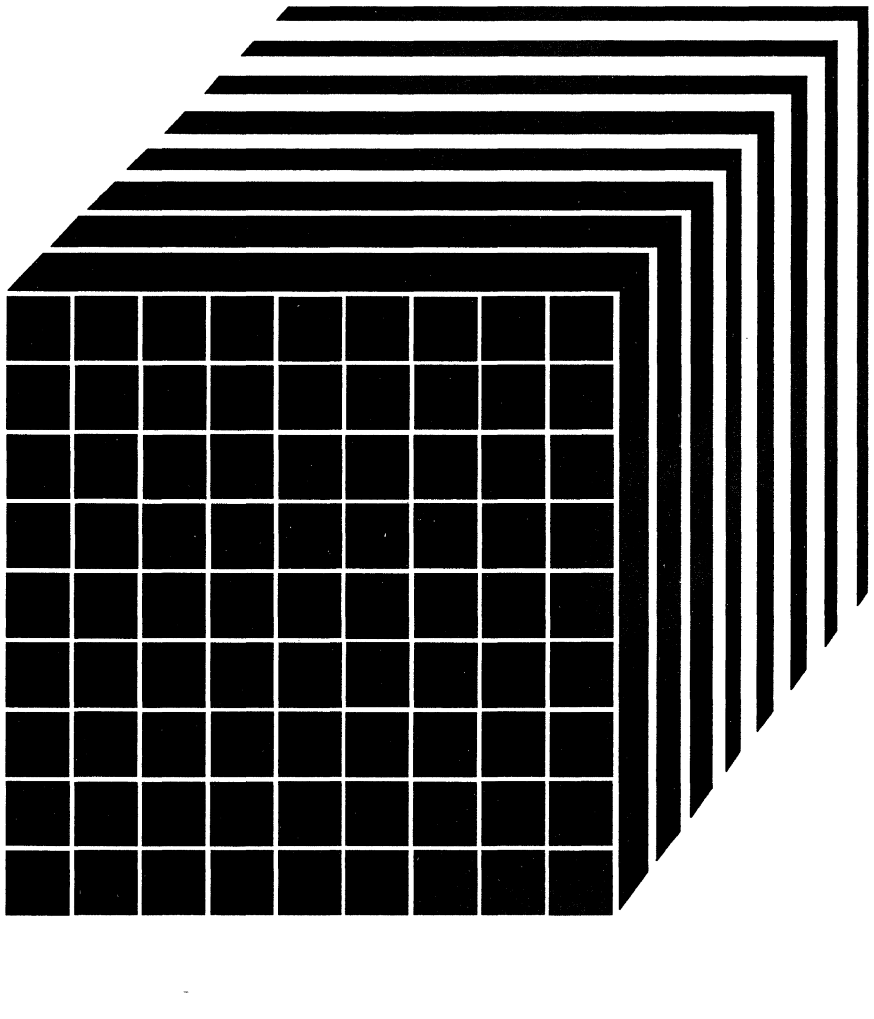
IBM

Virtual Machine/
System Product

System Logic and Problem Determination Guide Volume 1 (CP)

Release 5

LY20-0892-4



| **Fifth Edition (December 1986)**

| This edition, LY20-0892-4, is a major revision of LY20-0892-3 and applies to Release 5 of the Virtual Machine/System Product (VM/SP), Program Number 5664-167, and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Summary of Changes

A cumulative **Summary of Changes** begins on page 435.

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

Ordering Publications

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

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

Preface

This publication provides the IBM system hardware and software support personnel with the information needed to analyze problems that may occur on the IBM Virtual Machine/System Product (VM/SP).

How This Manual is Organized

This manual is one of two volumes:

- Volume 1. VM/SP Control Program (CP)
- Volume 2. VM/SP Conversational Monitor System (CMS).

Each volume contains logic description for the designated components of VM/SP. Each of these volumes is divided into four sections: Introduction, Method of Operation and Program Organization, Directory, and Diagnostic Aids.

The introduction describes VM/SP, program states, how to use processor resources, functional information, performance guidelines to follow, and CP interruption handling.

The method of operation and program organization sections contain the functions and relationships of the program routines in VM/SP. They indicate the program operation and organization in a general way to serve as a guide in understanding VM/SP. They are not meant to be a detailed analysis of VM/SP programming and cannot be used as such.

The directory contains a table of the CP modules and their entry points.

The diagnostic aids sections contain additional information useful for determining the cause of a problem.

Appendix A contains a description of VM/370 Extended Control-Program Support (ECPS: VM/370) that is used with VM/SP.

Appendix B describes VM/SP support for the IBM 3850 Mass Storage System (MSS).

Appendix C discusses MVS/System Extensions and MVS/System Product support.

How To Use This Manual

- Isolate the component of VM/SP in which the problem occurred.
- Use the list of restrictions in *VM/SP System Messages and Codes* to be certain that the operation that was being performed was valid.
- Use the directories and use the *VM/SP Data Areas and Control Block Logic Volume 1 (CP)* to help you to isolate the problem.
- Use the method of operation and program organization sections, if necessary, to understand the operation that was being performed.

Contents

Figures

Part 1: CP Introduction

This part contains the following information:

- VM/SP
- Program States
- Using Processor Resources
- Functional Information
- Performance Guidelines
- CP Interruption Handling.

Chapter 1. VM/SP

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 a single processor IBM System/370 system complex.

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 input/output (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 of 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 mode or multiprocessor mode. For a description of the additional processing performed when in attached processor mode, see "The Attached Processor and Multiprocessor Environments" on page 226.

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 I/O 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
- 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 an analogous 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 utilization 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 I/O 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 I/O 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 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 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/SP, or VM/370, 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 utilization. Further, a page can be brought into any available page frame; the necessary relocation is done during program execution by a combination of CP page management 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.

Inactive pages are kept on a direct access storage device (DASD). 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 in conjunction with 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 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 storage of the real machine to the virtual machine. CP will dynamically build 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). 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 self-modifying channel programs (excepting those of OS ISAM and OS/VS TCAM Level 5) are to execute under control of CP. For additional information on the virtual=real option, see *VM/SP CP for System Programming.*

Virtual Storage Preservation

Virtual storage preservation support is designed to preserve the contents of designated virtual machines if the system operator forces such a machine off the system, if CP abnormally terminates it, or if CP itself abnormally terminates.

At VM/SP system generation time, the user can specify which virtual machines are to be saved. The contents of these virtual machines are written out and saved in DASD space that must be previously allocated during system generation; the sequence in which virtual machines are saved can also be established. If a sequence for saving systems is not defined, then the systems are saved in the order in which virtual storage preservation was invoked for each. After the user logs on to the system again, the saved DASD area is restored by issuing the Initial Program Load (IPL) command, specifying the name of the defined DASD area. System generation parameters also allow another designated user to IPL the named SAVESYS area.

Either the VMSAVE directory option or the SET VMSAVE command may be used for saving the contents of a specific virtual machine. The VMSAVE facility can be nullified by SET VMSAVE OFF, SYSTEM CLEAR, DEFINE STORAGE, or normal LOGOFF.

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.

You can specify multiple VMSAVE target areas (areas in which the virtual machine is to be saved) 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 IPL command cannot refer to the area until a recovery operation has taken place. 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. Until the user releases that area, no other user can use it as a VMSAVE target area.

For more information on the VMSAVE facility, refer to *VM/SP CP for System Programming.*

Virtual Machine I/O Management

To decrease the size of the directory, commonly used control statements can be contained in a directory profile. The profile is defined by a PROFILE control statement, and each user entry may reference this profile via an INCLUDE control statement. The advantage gained is that these commonly used statements are defined only once (in the profile), instead of several times (in each user's entry).

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 must 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 I/O 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 topic "Reserve/Release" on page 122, located under "Scheduling I/O Requests" on page 118.

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

In attached processor or multiprocessor environments virtual I/O can be initiated by either processor; in attached processor systems 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 VM/SP multiprocessor system, real I/O can be handled by both processors as both processors have I/O capability.

In addition, the interrupts caused by the I/O operation (including channel errors) are reflected to the virtual machine for its interpretation and processing. If I/O 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).

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

See Appendix B, "VM/SP MSS Support" on page 409 of this volume for an explanation of additional processing when the virtual I/O request results in a real I/O request to a Mass Storage System (MSS) 3330V volume.

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 I/O 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 exhausted, and the virtual unit record devices appear in a not-ready status. The spooling operator may make additional spooling space available by using the class D SPTAPE command to dump output spool files to tape. He can also use the SPTAPE command to retrieve those files from the tape for output processing when spooling space requirements are not critical. See the description of the SPTAPE command in the *VM/SP CP Command Reference* for further information. In an extreme situation, the system operator may make additional spooling space available by purging existing spool files or by assigning additional direct access storage space to the spooling function.

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.

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, and defining spooling forms, classes, and destinations 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, both the Remote Spooling Communications Subsystem and the Remote Spooling Communications Subsystem Networking program product expect to find destination identification in the file tag, whenever an output spool file is submitted to them for transmission to a remote location. 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, except for fullscreen output, 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)
- 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:

START ((WARM | CKPT | FORCE | COLD)(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 log message data to the warm start area on an auxiliary DASD. 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 checkpoint area on an auxiliary DASD as they occur. 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 log message data 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. Each CP command is defined by a COMMD macro entry in module DMKCFC. Entries for logged-on users are placed beyond label COMNBEG1. Module DMKCMD also contains COMMD macro entries for subcommands. The COMMD macro has parameters defining command or subcommand name, class, type, entry point label, and the label of valid subcommands in DMKCMD.

Virtual machine execution can be stopped at any time by use of the terminal's attention key (for 3066 and 3270 terminals, the ENTER key is used); it 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 program status word (PSW) and the channel status word (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 IBM-supplied 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.*

You can extend the eight IBM-defined classes to up to 32 classes by adding an optional Class Control statement to the Directory. See the *VM/SP CP for System Programming* and the *VM/SP Planning Guide and Reference* for more information.

CP Messages

CP dynamically builds some of its messages in modules; these messages are then issued directly from the modules. However, CP issues many of its messages from a repository file.

CP dynamically creates a table for each language that a user sets. These tables determine the page of the message repository that contains the text associated with a particular message. Each table contains the range of message-ids on each page of the repository and the corresponding virtual address for that page.

The LANGBLOK contains the message table for a particular language. One LANGBLOK exists for each language, and these are chained together.

A user's VMBLOK points to the LANGBLOK for the language set for the user's virtual machine session. Changing a language means changing the pointer to the LANGBLOK in the user's VMBLOK (the VMLANG field).

During system initialization, CP creates the LANGBLOK for the installation default language's message repository. This repository is part of the CP nucleus, so virtual storage has already been allocated from the SYSTEM VMBLOK for the pages of the repository. The installation default language is always the first LANGBLOK in the LANGBLOK chain.

To handle an additional message repository for another language, CP creates a new VMBLOK through the virtual buffer manager. This VMBLOK has the userid LANGUAGE; it allows CP to access enough virtual storage for the message repository pages.

Unlike the repository in the nucleus, message repositories defined in the System Name Table (DMKSNT) must have virtual storage specifically allocated. When building the LANGBLOK for a message repository defined in the System Name Table (DMKSNT), CP allocates a virtual page from the LANGUAGE VMBLOK for each page of the message repository on DASD. CP then saves the virtual addresses in the LANGBLOK and initializes the swap table entry for each virtual page using the DASD address indicated in DMKSNT.

When a message is to be issued for a user, DMKERM finds the appropriate LANGBLOK; it then uses the table in LANGBLOK to get the virtual address of the repository page containing the message text. For the installation default language, CP uses information in the SYSTEM VMBLOK's page and swap tables to bring messages into real storage. For other supported languages, CP uses information in the LANGUAGE VMBLOK's page and swap tables to bring messages into real storage.

The CP Message Repository

The message compiler, the GENMSG command, generates a pageable text file from a message repository source file. The message repository text file consists of a:

- Single general information page
- Variable number of data pages.

Information Page: The first page of a repository contains general information about the repository. This information page is made up of an identifier, a header, and table entries. CP uses the information on this page to build various control blocks.

Figure 1 on page 13 shows the layout of the information page.

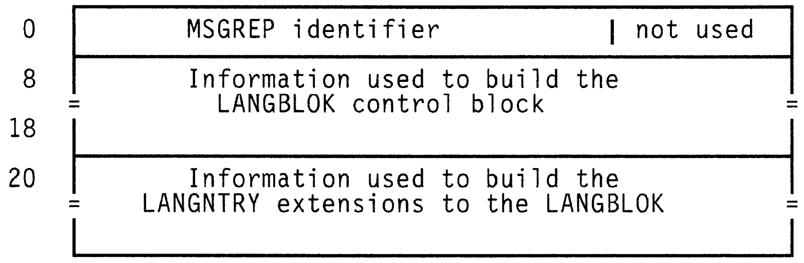

0
8
18
20
MSGREP identifier
not used
Information used to build the LANGBLOK control block
Information used to build the LANGNTRY extensions to the LANGBLOK

Figure 1. Information Page for a Message Repository

- Identifier

  The compiler sets the first six bytes of the information page to "MSGREP." This identifies the file as a message repository. The next two bytes are unused.

- Header

  This information is used to build the control block for a language, LANGBLOK. The compiler initializes this portion of the repository as follows:

Label	**Value**
LANGNEXT	0
LANGPAGE	# of data pages in the file (which corresponds to the # of table entries on the information page).
LANGFLAG	x'80' for a DBCS repository x'00' otherwise
LANGLOCK	0
LANGLANG	5 character langid, left justified, padded with blanks
LANGVMBK	0
all reserved fields	0

  CP initializes or modifies LANGNEXT, LANGFLAG, LANGLOCK, and LANGVMBK as appropriate when it builds the LANGBLOK control block for this repository.

- Table entries

  These are used to build LANGNTRY extension on the LANGBLOK. The compiler generates one table entry for each data page in the repository; CP then builds one LANGNTRY extension to the LANGBLOK for each table entry on this page.

  The compiler initializes each table entry as follows:

Label	Value
LANGLOW	message-id of the lowest number stored on the data page
LANGHIGH	message-id of the highest number stored on the data page
LANGADDR	0

CP initializes LANGADDR when it builds the LANGNTRY extension.

Data Pages: All pages following the repository page are data pages. Data pages contain CP error messages and responses. DMKERM references these data pages in order to display CP messages. Each data page consists of a header, message entries, and index entries. (Refer to Figure 2.)

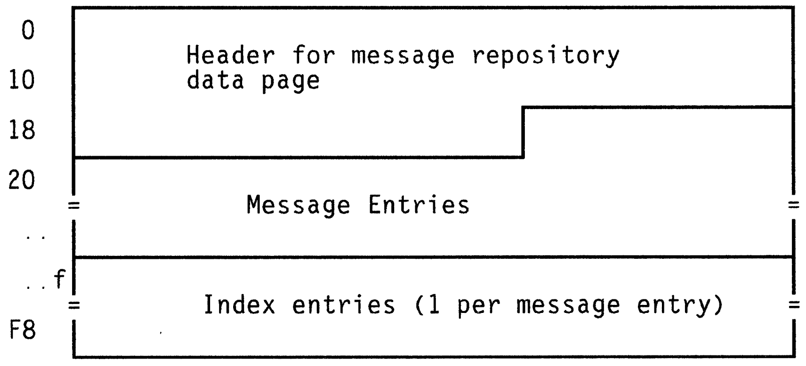

0
10
18
20
..
..f
F8
Header for message repository data page
Message Entries
Index entries (1 per message entry)

Figure 2. Data Page for a Message Repository

- Header

  The header on repository data pages is mapped by REPHEAD DSECT in module DMKERM

Label	Value
REPREP	string identifying the page as part of the message repository.
REPLANG	string indicating the language. This should be 5 bytes, with the language left justified and padded with blanks.

REPAPPL	string indicating this is a CP repository. The compiler sets this to DMK for a CP message repository.
REPSUB	substitution character that was specified in the message repository source file.
REPHDRLN	number of digits to display in the error message. (It is specified in the repository source file; however, CP ignores this value and always displays a 3 digit message number.)
REPCNT	number of message entries contained on this data page.
REPINXPT	displacement from the start of the 4K page to the index portion of the data page.
REPTXTPT	displacement from the start of the 4K page to the text of the first message on this data page.
REPDBCS	indicates whether the repository is DBCS (x'80') or not (x'00').

- Message entries

  Message entries are grouped together and follow immediately after the header. Message entries are variable in length; they consist of:

  - An action character (1 byte)
  - A length field, which indicates the length of the message text (1 byte)
  - Message text.

- Index entries

  Index entries on repository data pages are grouped together and follow the message entries. There is one index entry for each message entry on the data page. Index entries are mapped by the MSGINDEX DSECT in module DMKERM. They consist of:

  - A message identifier
    - Message number (2 bytes)
    - Format number (1 byte)
    - Line number (1 byte).

    For example, X'00050201' is message 5, format 2, line 1.

  - The displacement from the start of the 4K page to the message entry on the data page.

Module DMKERM

Many CP modules call DMKERM to display error messages. Module DMKERM builds the message-id from the input parameters and gets the address of the LANGBLOK that identifies the message repository used for the virtual machine. (The VMBLOK contains this LANGBLOK address.)

DMKERM then:

- Scans the LANGNTRY extensions to determine which data page of the message repository has the message to be displayed
- Pages in the repository data page containing the message
- Does a binary search on the index to find the index entry for the message to be displayed
- Gets the displacement to the message from the index entry
- Calculates the address of the message
- Sets up the message to be displayed.

Note: The message repository object file maintains 4K page boundaries:

- *Message texts do not cross page boundaries.*
- *All lines of a multiple line message are on the same 4K page.*
- *All formats for a given message do not have to be on the same 4K page.*

Chapter 2. 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:

- Causes an interruption that is handled by the Control Program or,
- 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 will execute on the virtual machine in a manner identical to its execution on a real System/370 processor, as long as it does not violate the CP restrictions. See *VM/SP System Messages and Codes* for a list of the restrictions.

Chapter 3. 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:

- Queue 1 (Q1)
- Queue 2 (Q2)
- Queue 3 (Q3).

In order to be dispatched from one of these queues, 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 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 in the dispatch list by their deadline priorities. 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 virtual machine receives its next time slice. A particular virtual machine's deadline priority for Q1 will be better (earlier) than its corresponding priority for Q2.

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 in Q2 are considered noninteractive. Virtual machines are selected to enter Q2 from 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:

- Eligible list
- 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 size 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.

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.

Queue 3

Q3 is an extension of Q2 scheduling. It helps to distinguish between noninteractive 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 that 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 will perform 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.

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 Q3 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.

SET QDROP Option: If the SET QDROP *userid* command is issued for a virtual machine, queue drop processing occurs, and the virtual machine's pages are not scanned or placed on the FLUSHLIST.

The USERS operand of the QDROP command provides for the temporary propagation of the QDROP OFF status to any virtual machine communicating via VMCF or IUCV to a (server) virtual machine. The QDROP status for the served virtual machine remains in effect only while messages are outstanding between it and the server machine.

Chapter 4. Functional Information

The functional diagrams that follow describe the program logic associated with various control program functions. Not all CP functions are described. These functional diagrams are meant to describe the CP functions about which you may want more detailed information if you are debugging, modifying, or updating CP.

Figure 3 on page 24 and Figure 4 on page 25 describe the CP initialization process.

Figure 5 on page 26 and Figure 6 on page 27 describe the real and virtual I/O control blocks used by CP in its I/O control.

Figure 7 on page 28, Figure 8 on page 29, and Figure 9 on page 30 show how CP handles SVC, external, and program interrupts.

The CP paging function is described in Figure 10 on page 31.

The CP spooling function (both virtual and real) is described in Figure 11 on page 32 and Figure 12 on page 33.

Figure 13 on page 34 shows how virtual tracing is performed.

Figure 14 on page 35 and Figure 15 on page 36 describe CP PER command processing and CP PER interrupt processing.

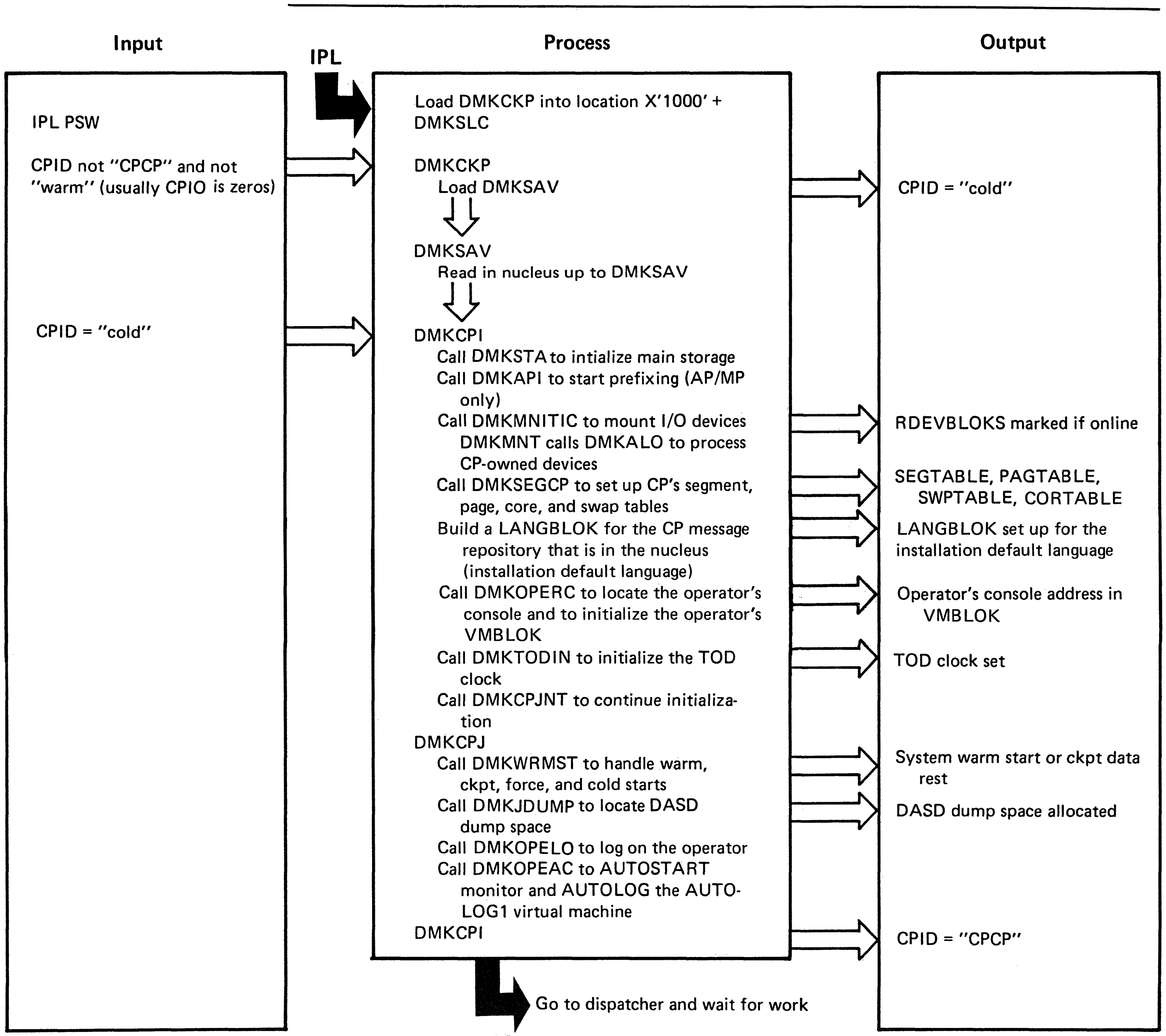
Figure 16 on page 37 shows the steps involved in translating a virtual address to a real address and gives an example of address translation.

Figure 17 on page 38 shows how SNA Consoles Communication Services (SNA CCS) communicates with the VTAM SNA Console Support (VM/VSCS) and with the rest of VM/SP.

Figure 18 on page 39 shows the structure of SNA Console Communication Services (SNA CCS) control blocks.

Figure 19 on page 40 shows the structure of SNA DIAL control blocks.

The functional information contained in these diagrams is intended for system programmers and IBM National Service Division program support representatives.



Input

IPL PSW

CPID not "CPCP" and not "warm" (usually CPIO is zeros)

CPID = "cold"

Process

IPL

Load DMKCKP into location X'1000' + DMKSLC

DMKCKP
  Load DMKSAV

DMKSAV
  Read in nucleus up to DMKSAV

DMKCPI
  Call DMKSTA to intialize main storage
  Call DMKAPI to start prefixing (AP/MP only)
  Call DMKMNITIC to mount I/O devices
    DMKMNT calls DMKALO to process CP-owned devices
  Call DMKSEGCP to set up CP's segment, page, core, and swap tables
  Build a LANGBLOK for the CP message repository that is in the nucleus (installation default language)
  Call DMKOPERC to locate the operator's console and to initialize the operator's VMBLOK
  Call DMKTODIN to initialize the TOD clock
  Call DMKCPJNT to continue initialization
DMKCPJ
  Call DMKWRMST to handle warm, ckpt, force, and cold starts
  Call DMKJDUMP to locate DASD dump space
  Call DMKOPELO to log on the operator
  Call DMKOPEAC to AUTOSTART monitor and AUTOLOG the AUTOLOG1 virtual machine
DMKCPI

Go to dispatcher and wait for work

Output

CPID = "cold"

RDEVBLOKS marked if online

SEGTABLE, PAGTABLE, SWPTABLE, CORTABLE

LANGBLOK set up for the installation default language

Operator's console address in VMBLOK

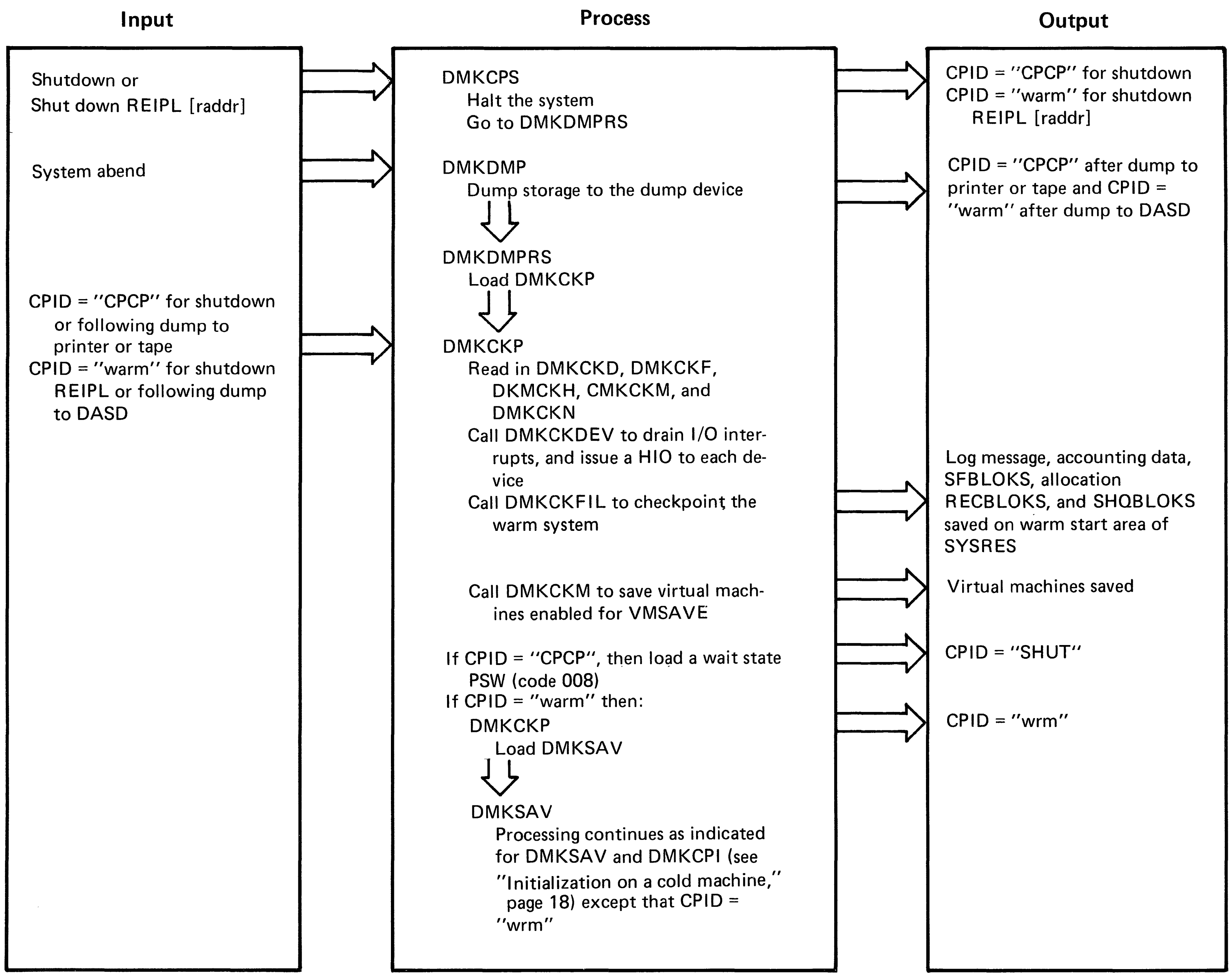
TOD clock set

System warm start or ckpt data rest

DASD dump space allocated

CPID = "CPCP"

Figure 3. Initialization on a Cold Machine



Input

Shutdown or
Shut down REIPL [raddr]

System abend

CPID = "CPCP" for shutdown
or following dump to
printer or tape
CPID = "warm" for shutdown
REIPL or following dump
to DASD

Process

DMKCPS
Halt the system
Go to DMKDMPRS

DMKDMP
Dump storage to the dump device

DMKDMPRS
Load DMKCKP

DMKCKP
Read in DMKCKD, DMKCKF,
DKMCKH, CMKCKM, and
DMKCKN
Call DMKCKDEV to drain I/O inter-
rupts, and issue a HIO to each de-
vice
Call DMKCKFIL to checkpoint the
warm system

Call DMKCKM to save virtual mach-
ines enabled for VMSAVE

If CPID = "CPCP", then load a wait state
PSW (code 008)
If CPID = "warm" then:
DMKCKP
Load DMKSAV

DMKSAV
Processing continues as indicated
for DMKSAV and DMKCPI (see
"Initialization on a cold machine,"
page 18) except that CPID =
"wrm"

Output

CPID = "CPCP" for shutdown
CPID = "warm" for shutdown
REIPL [raddr]

CPID = "CPCP" after dump to
printer or tape and CPID =
"warm" after dump to DASD

Log message, accounting data,
SFBLOKS, allocation
RECBLOKS, and SHQBLOKS
saved on warm start area of
SYSRES

Virtual machines saved

CPID = "SHUT"

CPID = "wrm"

Figure 4. System Shutdown and Automatic Warm Start

Figure 5. Real I/O Control Blocks

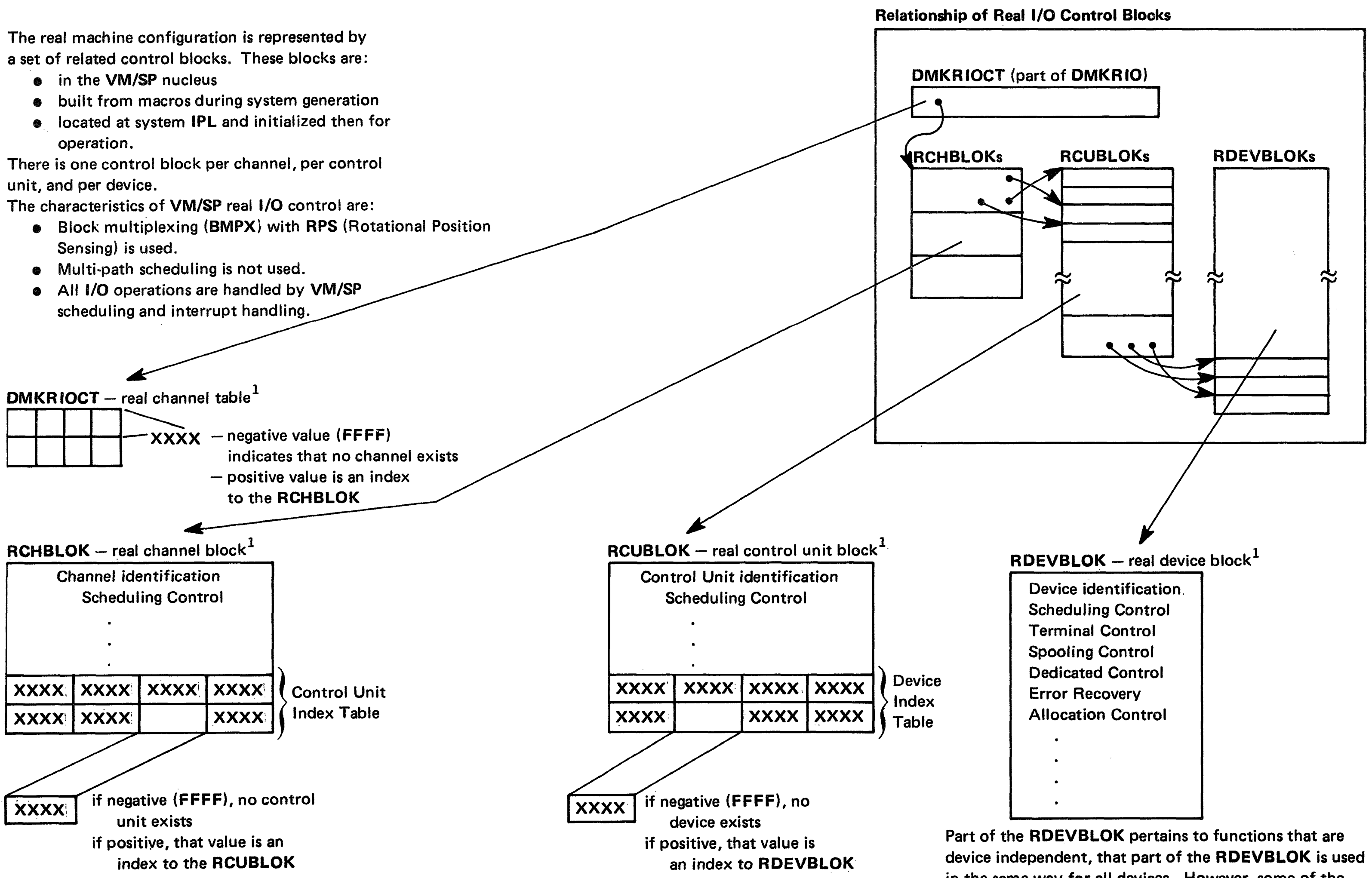
The real machine configuration is represented by a set of related control blocks. These blocks are:

- in the **VM/SP** nucleus
- built from macros during system generation
- located at system **IPL** and initialized then for operation.

There is one control block per channel, per control unit, and per device.

The characteristics of **VM/SP** real **I/O** control are:

- Block multiplexing (**BMPX**) with **RPS** (Rotational Position Sensing) is used.
- Multi-path scheduling is not used.
- All **I/O** operations are handled by **VM/SP** scheduling and interrupt handling.



Relationship of Real I/O Control Blocks
DMKRIOCT (part of DMKRIO)
RCHBLOKs
RCUBLOKs
RDEVBLOKs
DMKRIOCT – real channel table[1]
XXXX
– negative value (FFFF) indicates that no channel exists
– positive value is an index to the RCHBLOK
RCHBLOK – real channel block[1]
Channel identification
Scheduling Control
XXXX XXXX XXXX XXXX
XXXX XXXX XXXX
Control Unit Index Table
XXXX
if negative (FFFF), no control unit exists
if positive, that value is an index to the RCUBLOK
RCUBLOK – real control unit block[1]
Control Unit identification
Scheduling Control
XXXX XXXX XXXX XXXX
XXXX XXXX XXXX
Device Index Table
XXXX
if negative (FFFF), no device exists
if positive, that value is an index to RDEVBLOK
RDEVBLOK – real device block[1]
Device identification
Scheduling Control
Terminal Control
Spooling Control
Dedicated Control
Error Recovery
Allocation Control

Part of the **RDEVBLOK** pertains to functions that are device independent, that part of the **RDEVBLOK** is used in the same way for all devices. However, some of the fields in the **RDEVBLOK** have multiple uses, depending on the device type and function.

[1]For a complete description of CP control blocks, see *IBM Virtual Machine/System Product: Data Areas and Control Blocks*, Order No. LY20-0891.

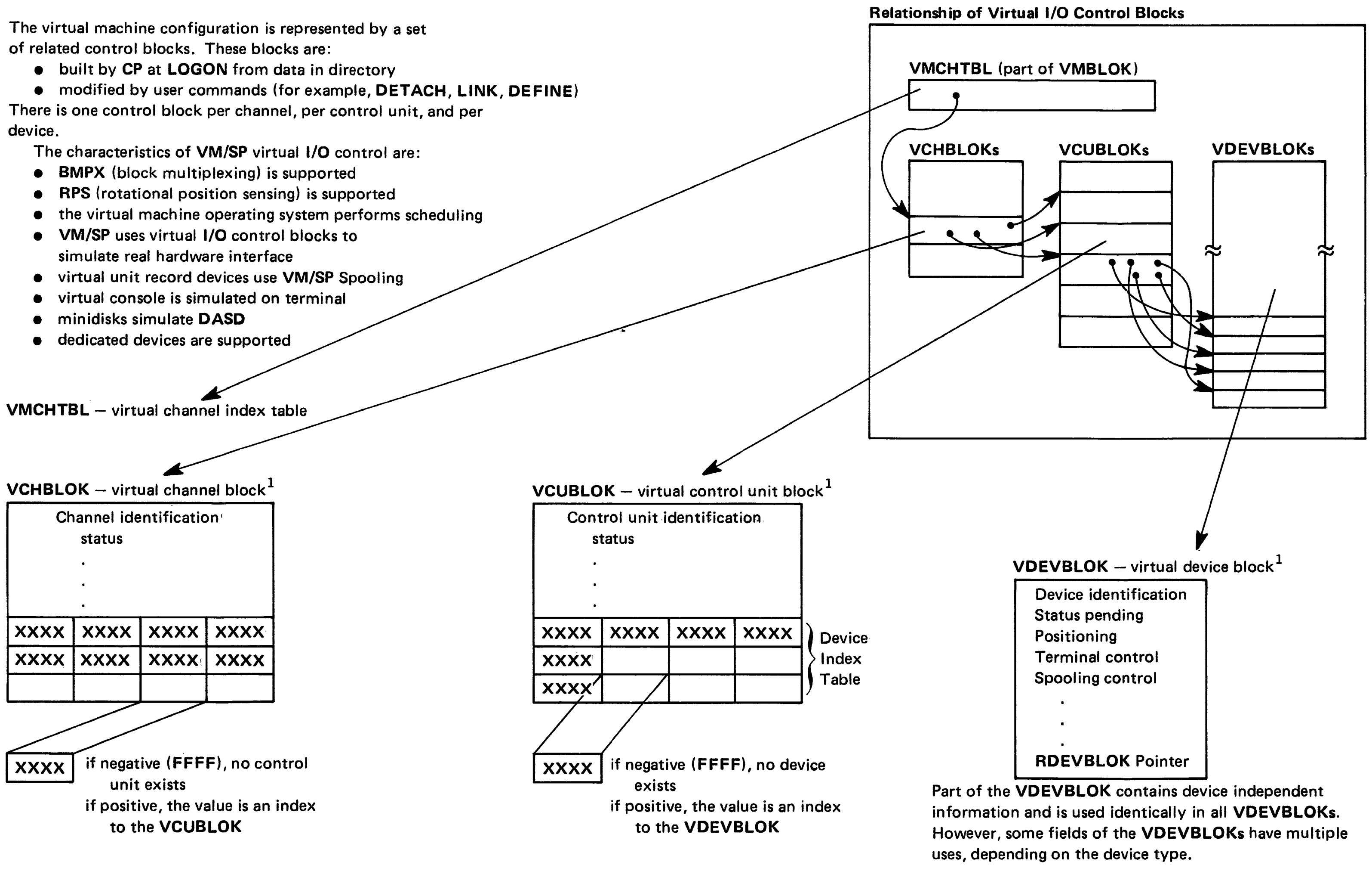
The virtual machine configuration is represented by a set of related control blocks. These blocks are:

- built by **CP** at **LOGON** from data in directory
- modified by user commands (for example, **DETACH**, **LINK**, **DEFINE**)

There is one control block per channel, per control unit, and per device.

The characteristics of **VM/SP** virtual **I/O** control are:

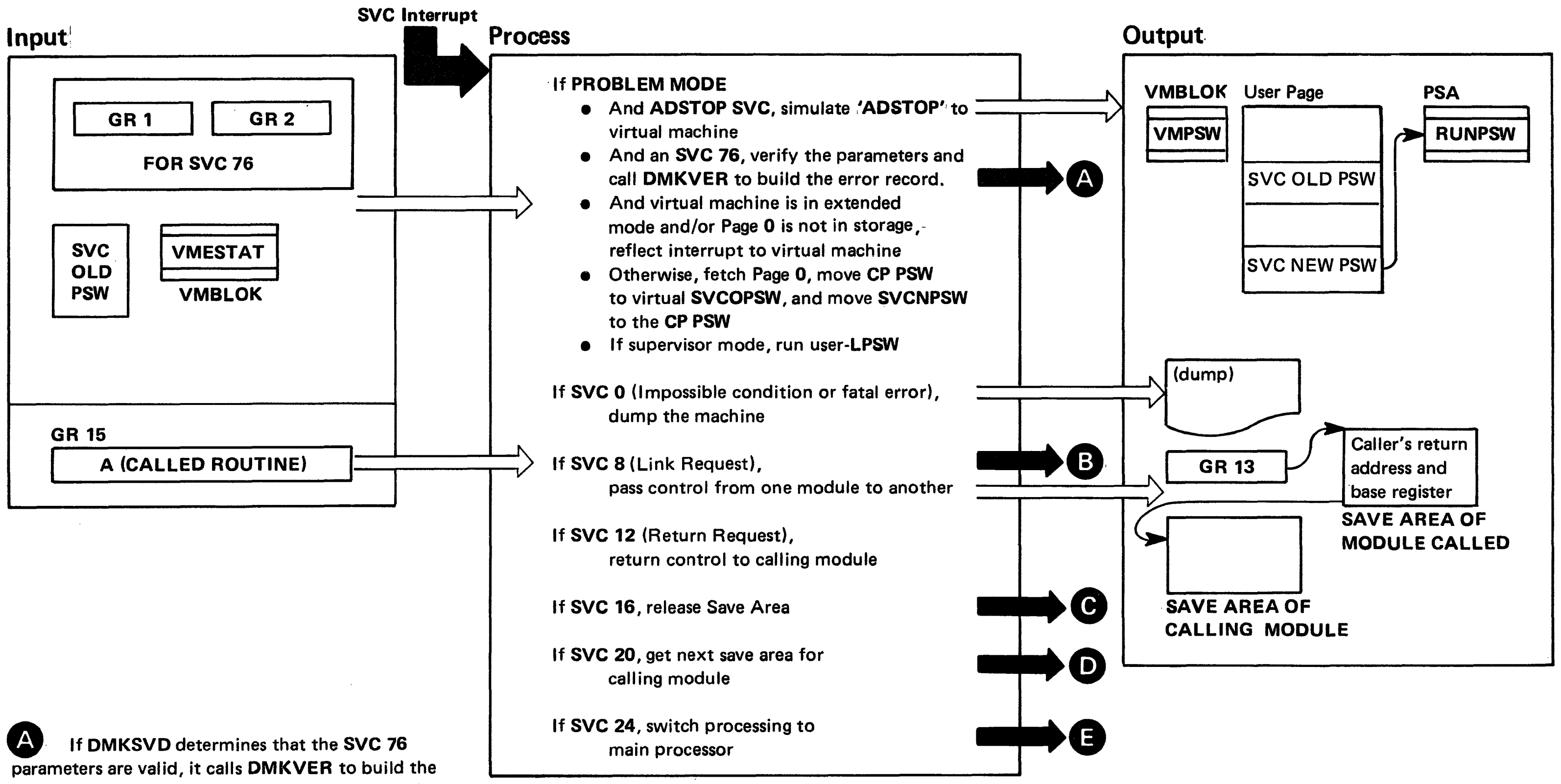
- **BMPX** (block multiplexing) is supported
- **RPS** (rotational position sensing) is supported
- the virtual machine operating system performs scheduling
- **VM/SP** uses virtual **I/O** control blocks to simulate real hardware interface
- virtual unit record devices use **VM/SP** Spooling
- virtual console is simulated on terminal
- minidisks simulate **DASD**
- dedicated devices are supported


Relationship of Virtual I/O Control Blocks
VMCHTBL (part of VMBLOK)
VCHBLOKs
VCUBLOKs
VDEVBLOKs
VMCHTBL – virtual channel index table
VCHBLOK – virtual channel block¹
Channel identification
status
XXXX XXXX XXXX XXXX
XXXX XXXX XXXX XXXX
XXXX
if negative (FFFF), no control unit exists
if positive, the value is an index to the VCUBLOK
VCUBLOK – virtual control unit block¹
Control unit identification
status
XXXX XXXX XXXX XXXX
XXXX
XXXX
Device Index Table
XXXX
if negative (FFFF), no device exists
if positive, the value is an index to the VDEVBLOK
VDEVBLOK – virtual device block¹
Device identification
Status pending
Positioning
Terminal control
Spooling control
RDEVBLOK Pointer

Part of the **VDEVBLOK** contains device independent information and is used identically in all **VDEVBLOKs**. However, some fields of the **VDEVBLOKs** have multiple uses, depending on the device type.

¹For a detailed description of the CP control blocks, see *IBM Virtual Machine /System Product Data Areas and Control Blocks,* Order No. LY20-0891.

Figure 6. Virtual I/O Control Blocks



If **PROBLEM MODE**

- And **ADSTOP SVC**, simulate **'ADSTOP'** to virtual machine
- And an **SVC 76**, verify the parameters and call **DMKVER** to build the error record.
- And virtual machine is in extended mode and/or Page **0** is not in storage, reflect interrupt to virtual machine
- Otherwise, fetch Page **0**, move **CP PSW** to virtual **SVCOPSW**, and move **SVCNPSW** to the **CP PSW**
- If supervisor mode, run user-**LPSW**

If **SVC 0** (Impossible condition or fatal error), dump the machine

If **SVC 8** (Link Request), pass control from one module to another

If **SVC 12** (Return Request), return control to calling module

If **SVC 16**, release Save Area

If **SVC 20**, get next save area for calling module

If **SVC 24**, switch processing to main processor

A If **DMKSVD** determines that the **SVC 76** parameters are valid, it calls **DMKVER** to build the error record. If the parameters are not valid or if **DMKVER** cannot build the error record, **DMKSVD** reflects the **SVC** back to the virtual machine. If the error record is recorded, **DMKVER** gives control to the dispatcher with the user's running status set to return to the next sequential instruction following the **SVC 76**.

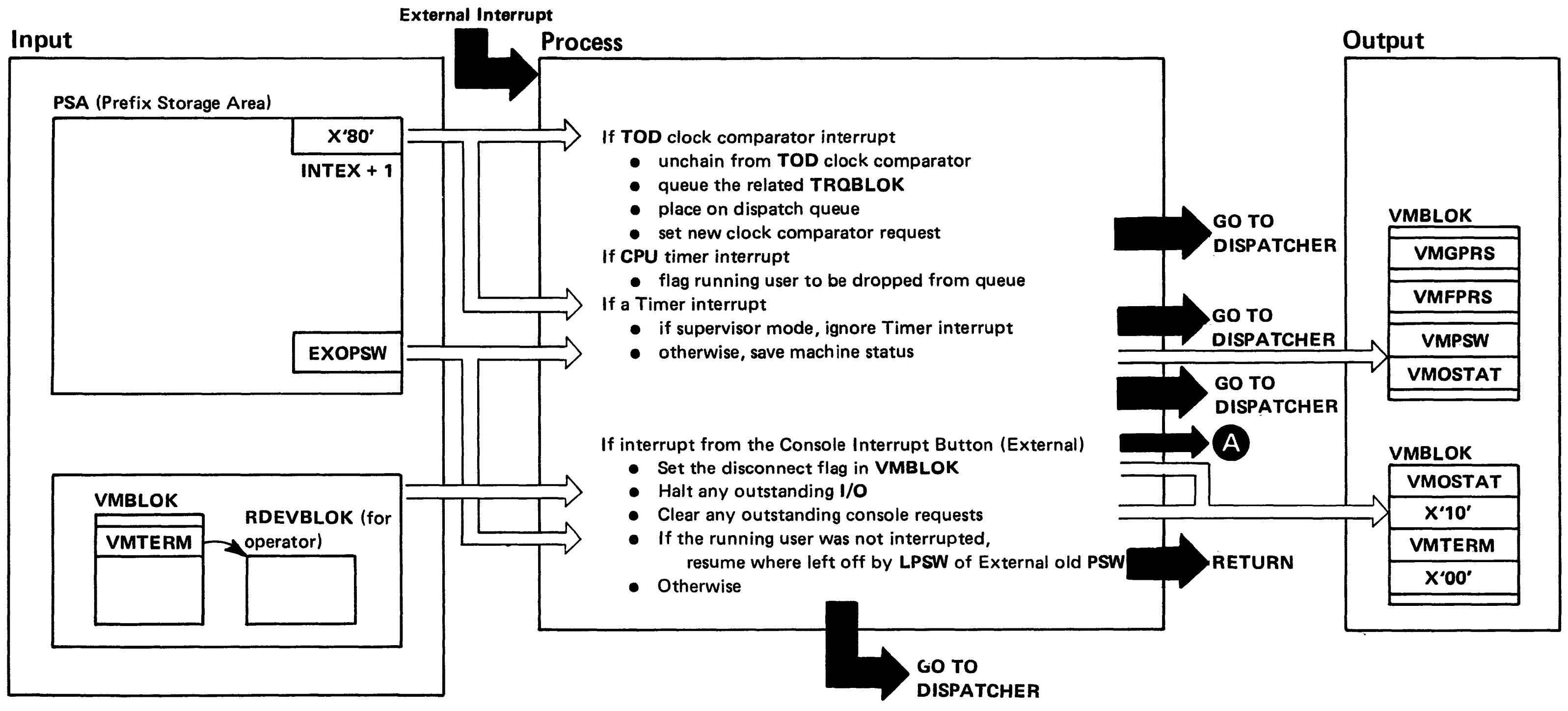
B A new save area is acquired and passed on. The caller's addressability register (**R 12**), the save area address (**R 13**), and the return address (**SVCOPSW**) are saved in the new save area.

C Control is returned to module issuing **SVC 16**, rather than to calling module as in **SVC 12**.

D Return is to module issuing **SVC 20**.

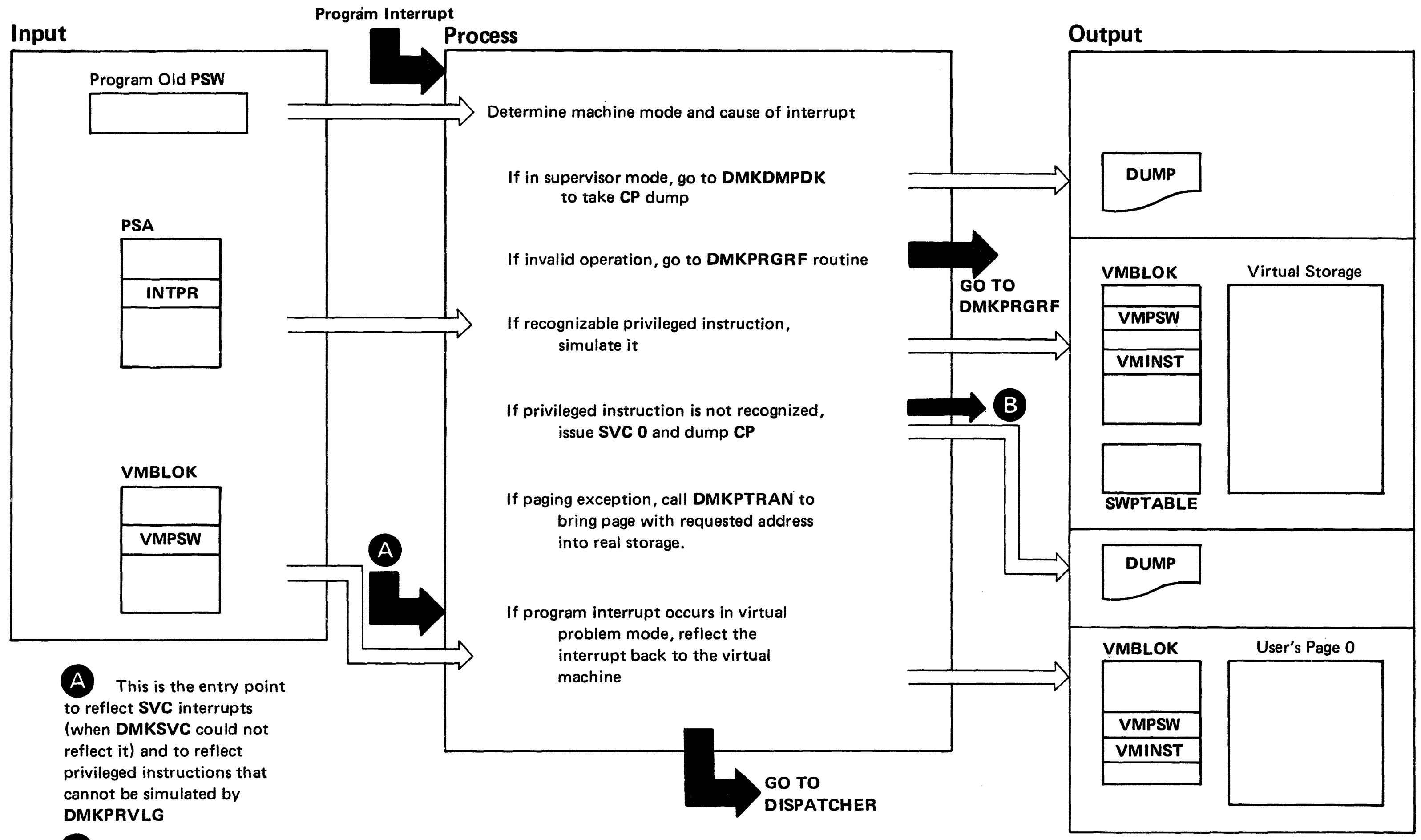
E Return is on other processor to module issuing **SVC 24**.

Figure 7. SVC Interrupt Handling


Input
External Interrupt
Process
Output
PSA (Prefix Storage Area)
X'80'
INTEX + 1
EXOPSW
VMBLOK
VMTERM
RDEVBLOK (for operator)
If TOD clock comparator interrupt
• unchain from TOD clock comparator
• queue the related TRQBLOK
• place on dispatch queue
• set new clock comparator request
If CPU timer interrupt
• flag running user to be dropped from queue
If a Timer interrupt
• if supervisor mode, ignore Timer interrupt
• otherwise, save machine status
If interrupt from the Console Interrupt Button (External)
• Set the disconnect flag in VMBLOK
• Halt any outstanding I/O
• Clear any outstanding console requests
• If the running user was not interrupted, resume where left off by LPSW of External old PSW
• Otherwise
GO TO DISPATCHER
GO TO DISPATCHER
GO TO DISPATCHER
A
RETURN
GO TO DISPATCHER
VMBLOK
VMGPRS
VMFPRS
VMPSW
VMOSTAT
VMBLOK
VMOSTAT
X'10'
VMTERM
X'00'

Ⓐ External interrupt from control panel is used to disconnect the system operator's terminal. The system operator may reconnect at any other terminal via the LOGON command.

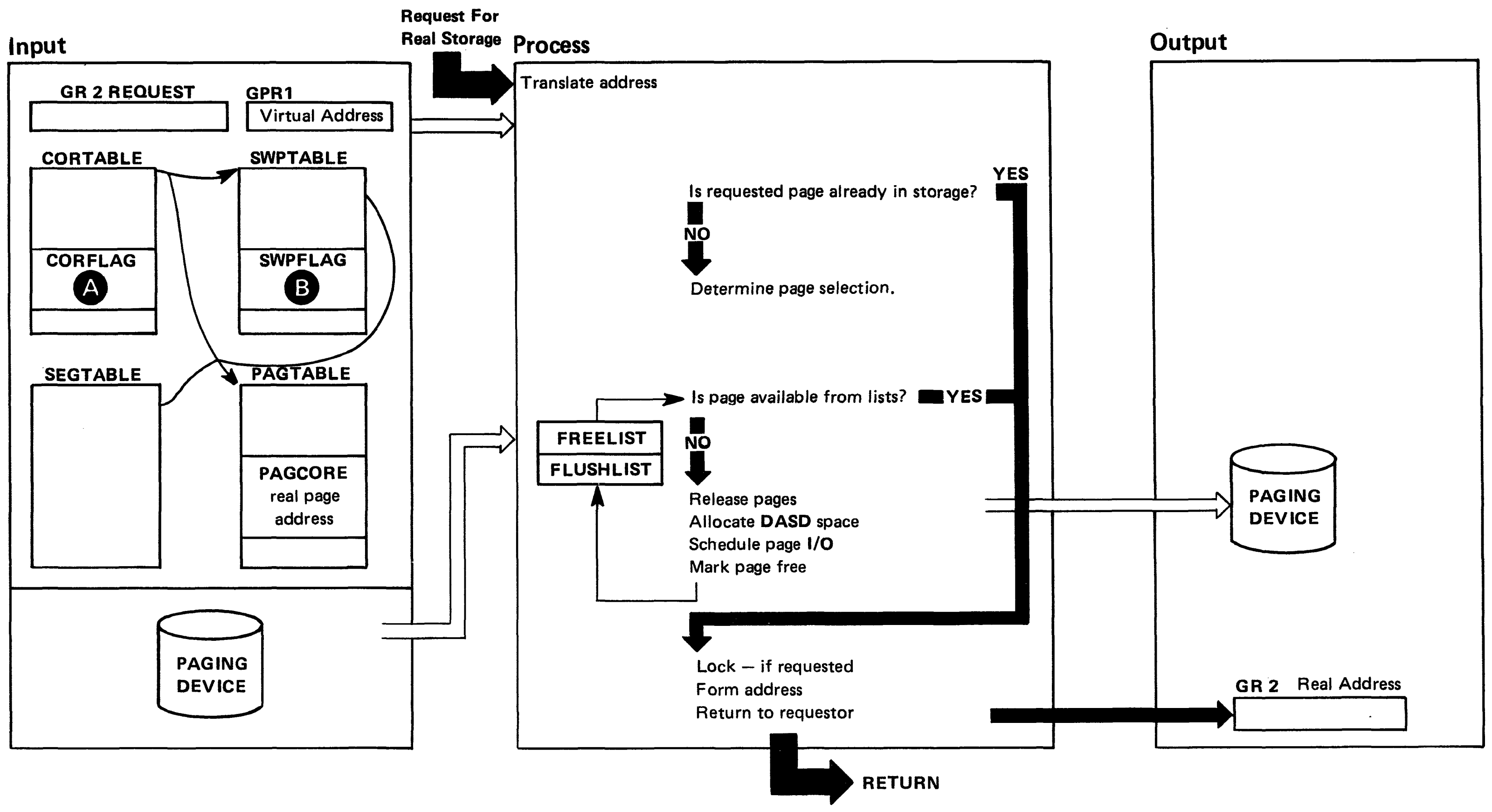
Figure 8. External Interrupt Handling


Input
Program Old PSW
PSA
INTPR
VMBLOK
VMPSW
Program Interrupt
Process
Determine machine mode and cause of interrupt
If in supervisor mode, go to DMKDMPDK to take CP dump
If invalid operation, go to DMKPRGRF routine
If recognizable privileged instruction, simulate it
If privileged instruction is not recognized, issue SVC 0 and dump CP
If paging exception, call DMKPTRAN to bring page with requested address into real storage.
If program interrupt occurs in virtual problem mode, reflect the interrupt back to the virtual machine
A
B
GO TO DMKPRGRF
GO TO DISPATCHER
Output
DUMP
VMBLOK
VMPSW
VMINST
Virtual Storage
SWPTABLE
DUMP
VMBLOK
User's Page 0
VMPSW
VMINST

A This is the entry point to reflect **SVC** interrupts (when **DMKSVC** could not reflect it) and to reflect privileged instructions that cannot be simulated by **DMKPRVLG**

B Invalid operation code is in **GR 0**. The **VMINST** field of the **VMBLOK** contains the image of the privileged instruction that caused the interrupt

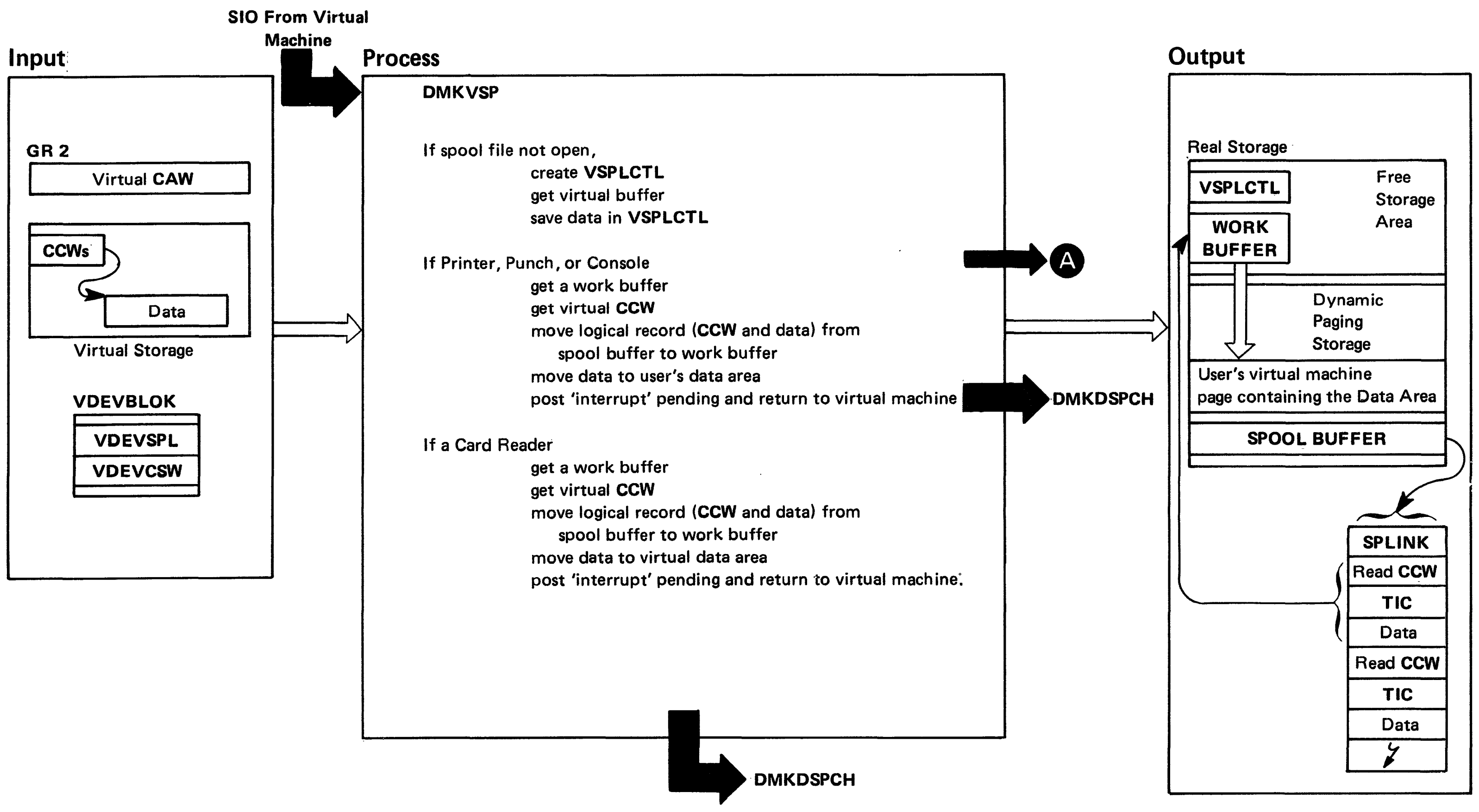
Figure 9. Program Interrupt Handling



A Bits defined for CORFLAG			
CORIOLCK	EQU	X'80'	Page locked for I/O
CORCFLCK	EQU	X'40'	Page locked by console function
CORFLUSH	EQU	X'20'	Page is in flush list
CORFREE	EQU	X'10'	Page is in free list
CORSHARE	EQU	X'08'	Page is shared
CORRSV	EQU	X'04'	Page is reserved
CORDISA	EQU	X'01'	Page disabled – not available

B Bits defined for SWPFLAG			
SWPTRANS	EQU	X'80'	Page in transit
SWPRECMP	EQU	X'40'	Page permanently assigned
SWPALLOC	EQU	X'20'	Page enqueued for allocation
SWPSHR	EQU	X'10'	Page shared
SWPREF1	EQU	X'08'	1st half page referenced
SWPCHG1	EQU	X'04'	1st half page changed
SWPREF2	EQU	X'02'	2nd half page referenced
SWPCHG2	EQU	X'01'	2nd half page changed

Figure 10. Paging



```
DMKVSP

If spool file not open,
        create VSPLCTL
        get virtual buffer
        save data in VSPLCTL

If Printer, Punch, or Console
        get a work buffer
        get virtual CCW
        move logical record (CCW and data) from
            spool buffer to work buffer
        move data to user's data area
        post 'interrupt' pending and return to virtual machine

If a Card Reader
        get a work buffer
        get virtual CCW
        move logical record (CCW and data) from
            spool buffer to work buffer
        move data to virtual data area
        post 'interrupt' pending and return to virtual machine.
```

Figure 11. Virtual Spooling

(A) Virtual console spooling is the same as printer spooling except that:

- A skip to channel one CCW is inserted every 60 lines of output
- The operator's virtual console spool buffer is written for every 16 lines of output
- The Virtual spool buffer is written to the allocated spool device when the first CCW is placed in the Virtual buffer. The buffer is kept in a pseudo closed state so that checkpoint saves the buffer in the event of a system failure.

Figure 12. Real Spooling

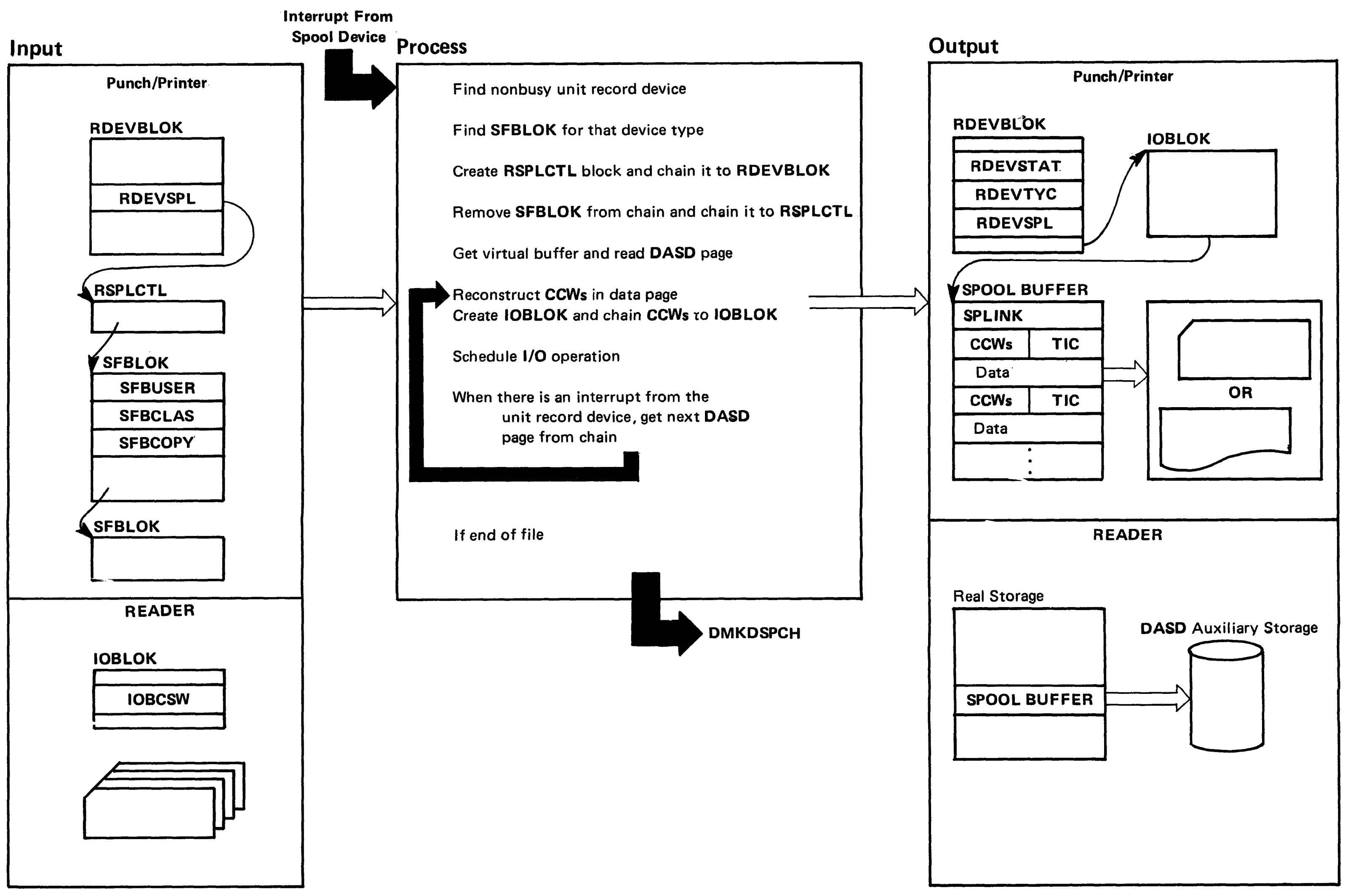
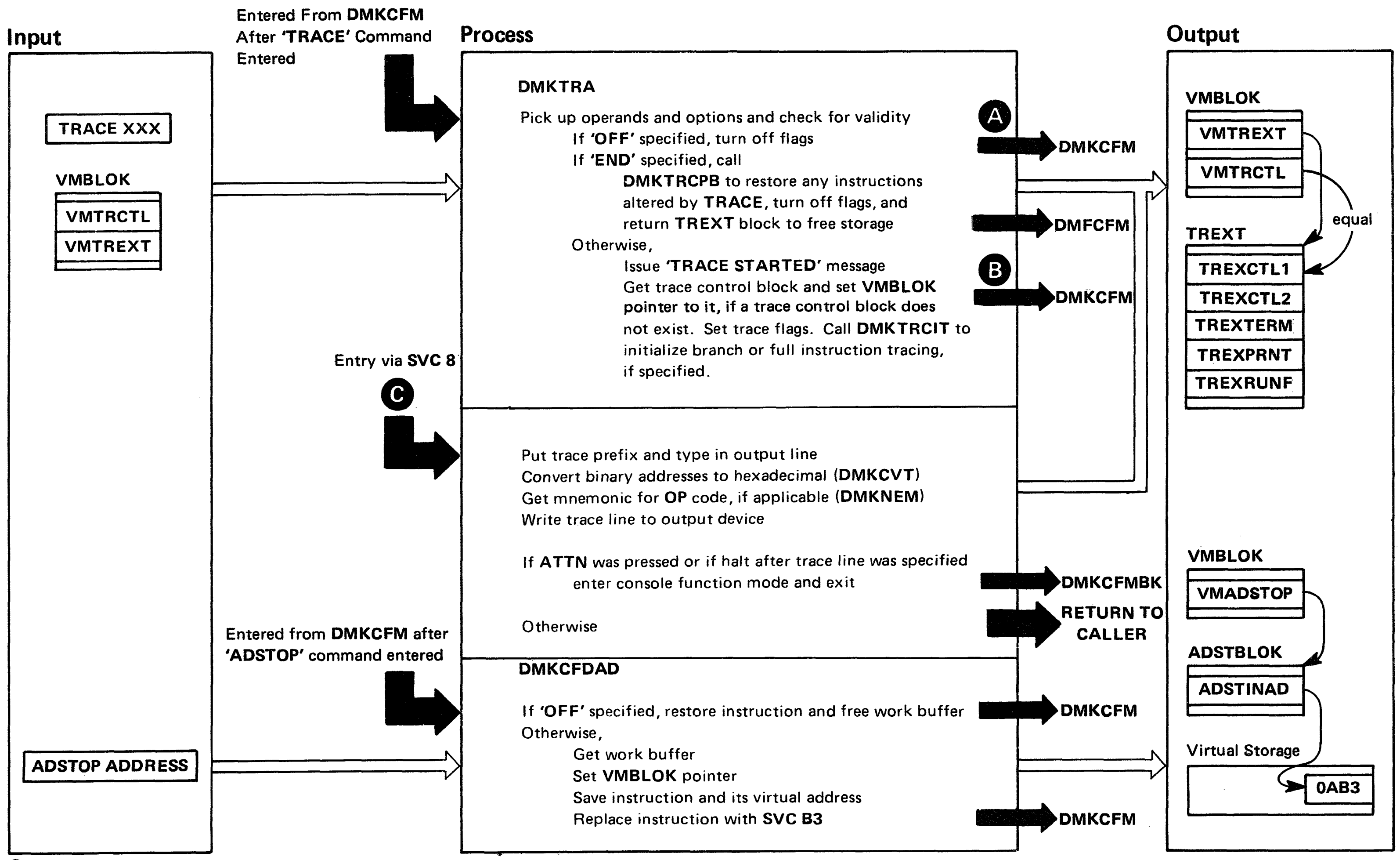

Input
Punch/Printer
RDEVBLOK
RDEVSPL
RSPLCTL
SFBLOK
SFBUSER
SFBCLAS
SFBCOPY
SFBLOK
READER
IOBLOK
IOBCSW
Interrupt From Spool Device
Process
Find nonbusy unit record device
Find SFBLOK for that device type
Create RSPLCTL block and chain it to RDEVBLOK
Remove SFBLOK from chain and chain it to RSPLCTL
Get virtual buffer and read DASD page
Reconstruct CCWs in data page
Create IOBLOK and chain CCWs to IOBLOK
Schedule I/O operation
When there is an interrupt from the unit record device, get next DASD page from chain
If end of file
DMKDSPCH
Output
Punch/Printer
RDEVBLOK
RDEVSTAT
RDEVTYC
RDEVSPL
IOBLOK
SPOOL BUFFER
SPLINK
CCWs
TIC
Data
CCWs
TIC
Data
OR
READER
Real Storage
SPOOL BUFFER
DASD Auxiliary Storage

Figure 13. Virtual Tracing


Input
TRACE XXX
VMBLOK
VMTRCTL
VMTREXT
ADSTOP ADDRESS
Entered From DMKCFM After 'TRACE' Command Entered
Entry via SVC 8
C
Entered from DMKCFM after 'ADSTOP' command entered
Process
DMKTRA
Pick up operands and options and check for validity
If 'OFF' specified, turn off flags
If 'END' specified, call DMKTRCPB to restore any instructions altered by TRACE, turn off flags, and return TREXT block to free storage
Otherwise,
Issue 'TRACE STARTED' message
Get trace control block and set VMBLOK pointer to it, if a trace control block does not exist. Set trace flags. Call DMKTRCIT to initialize branch or full instruction tracing, if specified.
A
DMKCFM
DMFCFM
B
DMKCFM
Put trace prefix and type in output line
Convert binary addresses to hexadecimal (DMKCVT)
Get mnemonic for OP code, if applicable (DMKNEM)
Write trace line to output device
If ATTN was pressed or if halt after trace line was specified enter console function mode and exit
DMKCFMBK
Otherwise
RETURN TO CALLER
DMKCFDAD
If 'OFF' specified, restore instruction and free work buffer
DMKCFM
Otherwise,
Get work buffer
Set VMBLOK pointer
Save instruction and its virtual address
Replace instruction with SVC B3
DMKCFM
Output
VMBLOK
VMTREXT
VMTRCTL
equal
TREXT
TREXCTL1
TREXCTL2
TREXTERM
TREXPRNT
TREXRUNF
VMBLOK
VMADSTOP
ADSTBLOK
ADSTINAD
Virtual Storage
0AB3

Comments

A If this turns off the last flag, then the TREXT block is returned to free storage. If branch and instruction tracing are both turned off, DMKTRCPB is called to put back any instructions altered by TRACE.

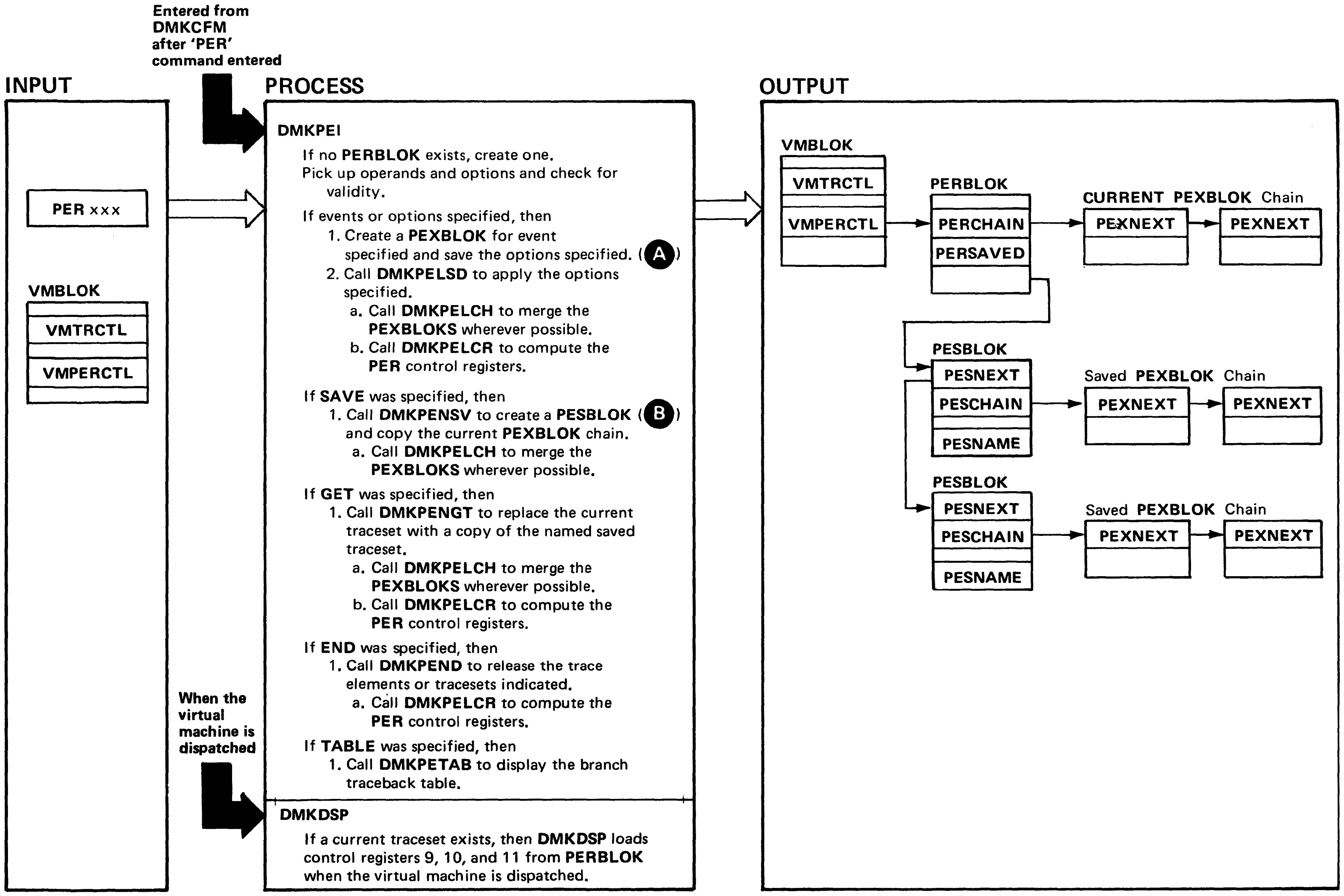
B VMTRCTL and TREXCTL1 are identical.

C Entry via SVC 8 as follows

	Entry Point	From
External Interrupt	DMKTRCEX	DMKDSP
I/O Interrupt	DMKTRCIO	DMKDSP
Program Interrupt	DMKTRCPG	DMKPRG
Privileged Instructions	DMKTRCPV	DMKPRV
I/O Operations	DMKTRDSI	DMKVIOEX
Virtual and Real CSWs	DMKTRCSW	DMKVIOIN

	Entry Point	From
SVC, branch or full instruction trace	DMKTRCSV	DMKPSA
Restore user instructions altered by tracing	DMKTRCPB	DMKTRA
Initialize instruction tracing	DMKTRCIT	DMKTRA

Figure 14. CP PER Command Processing



DMKPEI

If no **PERBLOK** exists, create one.
Pick up operands and options and check for validity.

If events or options specified, then
1. Create a **PEXBLOK** for event specified and save the options specified. (A)
2. Call **DMKPELSD** to apply the options specified.
   a. Call **DMKPELCH** to merge the **PEXBLOKS** wherever possible.
   b. Call **DMKPELCR** to compute the **PER** control registers.

If **SAVE** was specified, then
1. Call **DMKPENSV** to create a **PESBLOK** (B) and copy the current **PEXBLOK** chain.
   a. Call **DMKPELCH** to merge the **PEXBLOKS** wherever possible.

If **GET** was specified, then
1. Call **DMKPENGT** to replace the current traceset with a copy of the named saved traceset.
   a. Call **DMKPELCH** to merge the **PEXBLOKS** wherever possible.
   b. Call **DMKPELCR** to compute the **PER** control registers.

If **END** was specified, then
1. Call **DMKPEND** to release the trace elements or tracesets indicated.
   a. Call **DMKPELCR** to compute the **PER** control registers.

If **TABLE** was specified, then
1. Call **DMKPETAB** to display the branch traceback table.

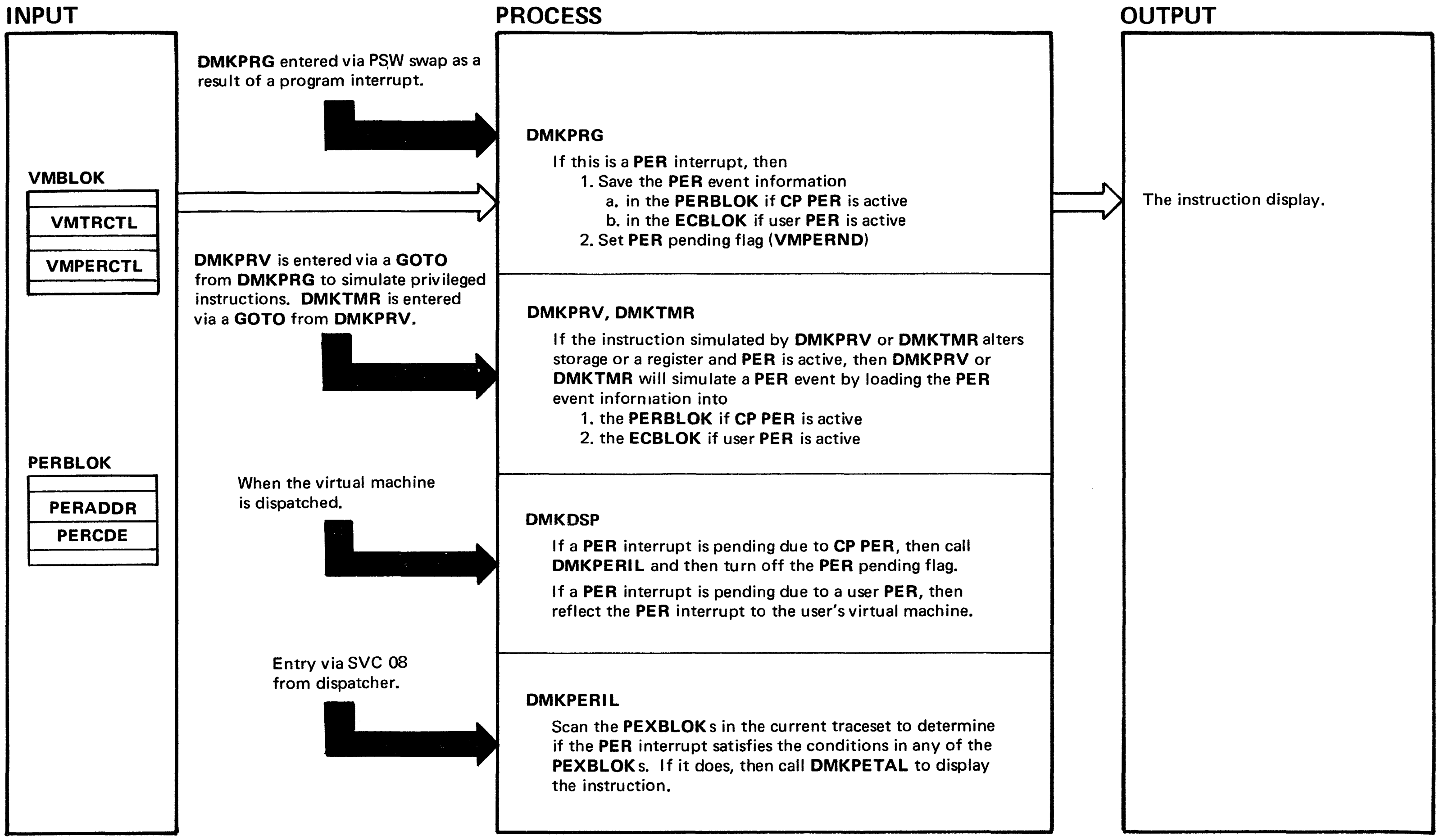
DMKDSP

If a current traceset exists, then **DMKDSP** loads control registers 9, 10, and 11 from **PERBLOK** when the virtual machine is dispatched.

Comments

(A) Each trace element is represented by a **PEXBLOK.**

(B) The **PEXBLOK**s for each saved traceset are chained from a **PESBLOK.**



DMKPRG entered via PSW swap as a result of a program interrupt.

DMKPRV is entered via a **GOTO** from **DMKPRG** to simulate privileged instructions. **DMKTMR** is entered via a **GOTO** from **DMKPRV.**

When the virtual machine is dispatched.

Entry via SVC 08 from dispatcher.

DMKPRG

If this is a **PER** interrupt, then
1. Save the **PER** event information
   a. in the **PERBLOK** if **CP PER** is active
   b. in the **ECBLOK** if user **PER** is active
2. Set **PER** pending flag (**VMPERND**)

DMKPRV, DMKTMR

If the instruction simulated by **DMKPRV** or **DMKTMR** alters storage or a register and **PER** is active, then **DMKPRV** or **DMKTMR** will simulate a **PER** event by loading the **PER** event information into
1. the **PERBLOK** if **CP PER** is active
2. the **ECBLOK** if user **PER** is active

DMKDSP

If a **PER** interrupt is pending due to **CP PER**, then call **DMKPERIL** and then turn off the **PER** pending flag.

If a **PER** interrupt is pending due to a user **PER**, then reflect the **PER** interrupt to the user's virtual machine.

DMKPERIL

Scan the **PEXBLOK**s in the current traceset to determine if the **PER** interrupt satisfies the conditions in any of the **PEXBLOK**s. If it does, then call **DMKPETAL** to display the instruction.

The instruction display.

Figure 15. CP PER Interrupt Processing

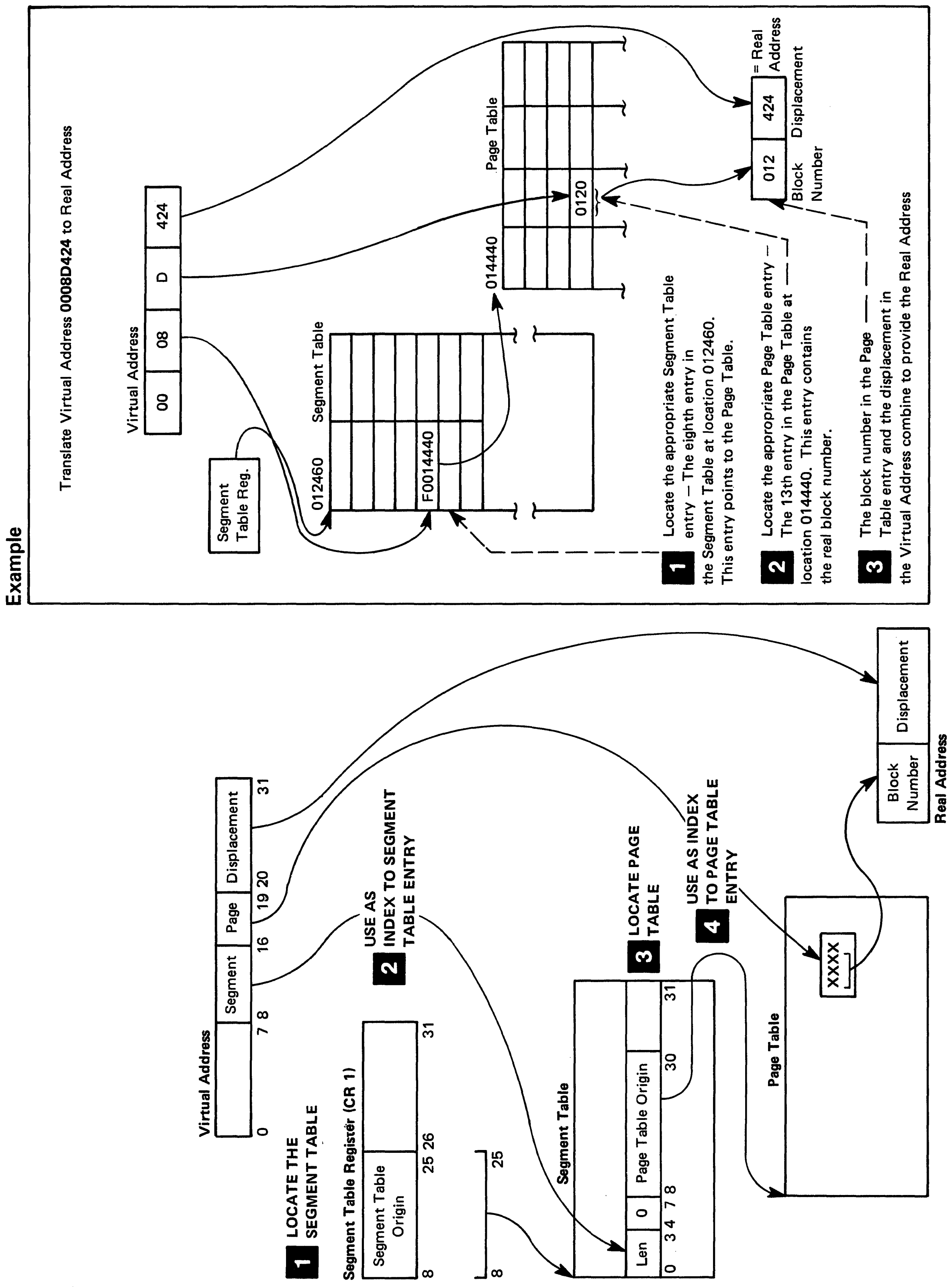

Example
Translate Virtual Address 0008D424 to Real Address
Virtual Address
00
08
D
424
Segment Table Reg.
012460
Segment Table
F0014440
014440
Page Table
0120
012
424
= Real Address
Block Number
Displacement
1 Locate the appropriate Segment Table entry – The eighth entry in the Segment Table at location 012460. This entry points to the Page Table.
2 Locate the appropriate Page Table entry – The 13th entry in the Page Table at location 014440. This entry contains the real block number.
3 The block number in the Page Table entry and the displacement in the Virtual Address combine to provide the Real Address
Virtual Address
Segment
Page
Displacement
0
7 8
16
19 20
31
1 LOCATE THE SEGMENT TABLE
Segment Table Register (CR 1)
Segment Table Origin
8
25 26
31
8
25
2 USE AS INDEX TO SEGMENT TABLE ENTRY
Segment Table
Len
0
Page Table Origin
0
3 4
7 8
30
31
3 LOCATE PAGE TABLE
4 USE AS INDEX TO PAGE TABLE ENTRY
Page Table
XXXX
Block Number
Displacement
Real Address

Figure 16. Virtual-to-Real Address Translation

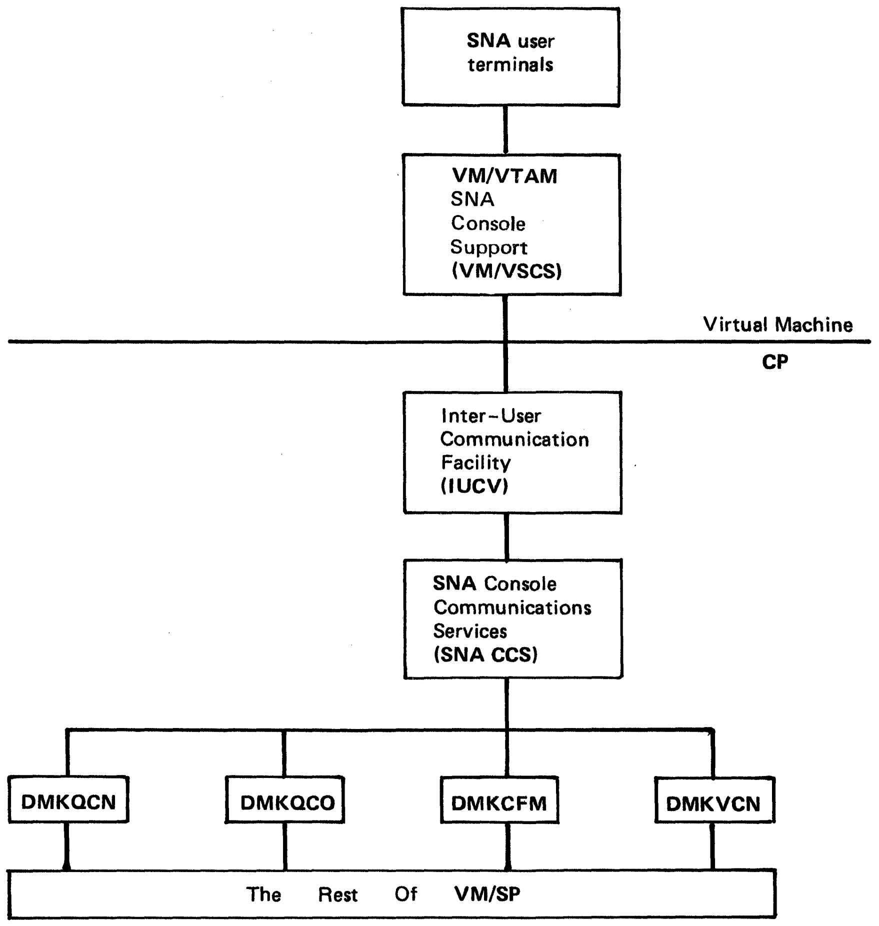

SNA user terminals
VM/VTAM SNA Console Support (VM/VSCS)
Virtual Machine
CP
Inter-User Communication Facility (IUCV)
SNA Console Communications Services (SNA CCS)
DMKQCN
DMKQCO
DMKCFM
DMKVCN
The Rest Of VM/SP

Figure 17. SNA CCS Interfaces

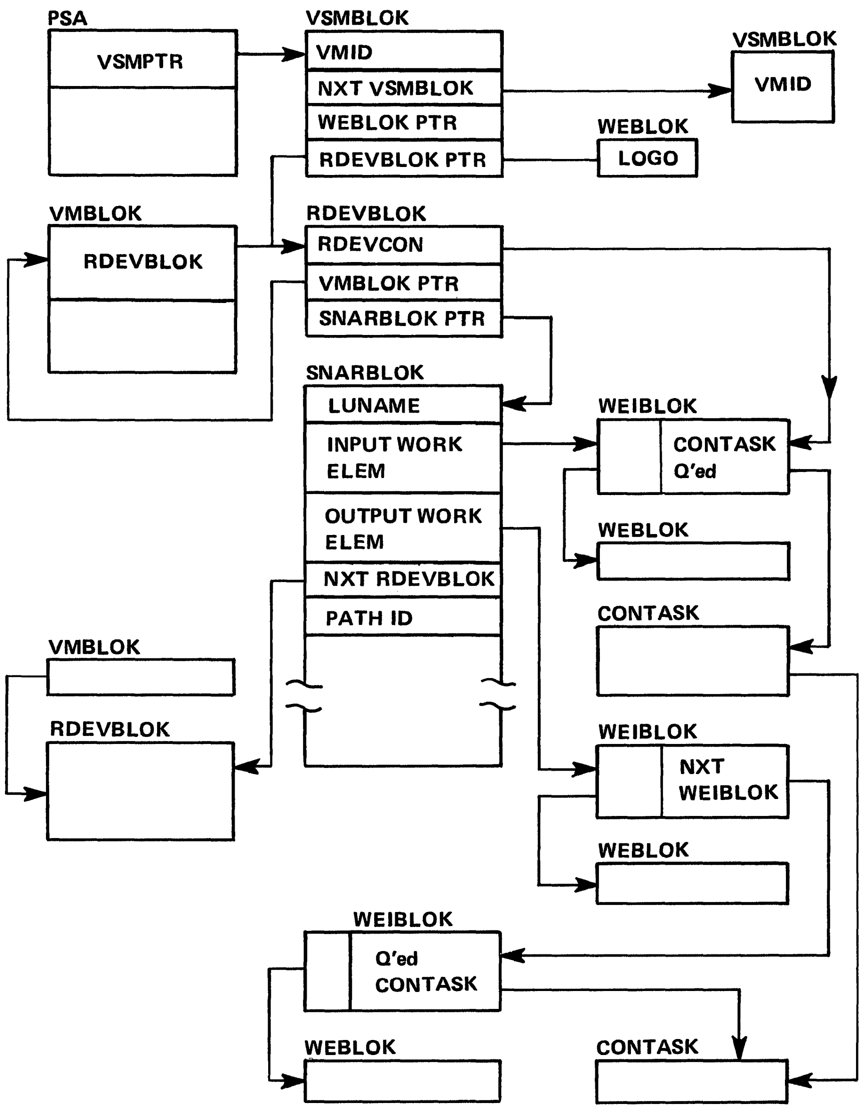


Figure 18. SNA CCS Control Block Structure

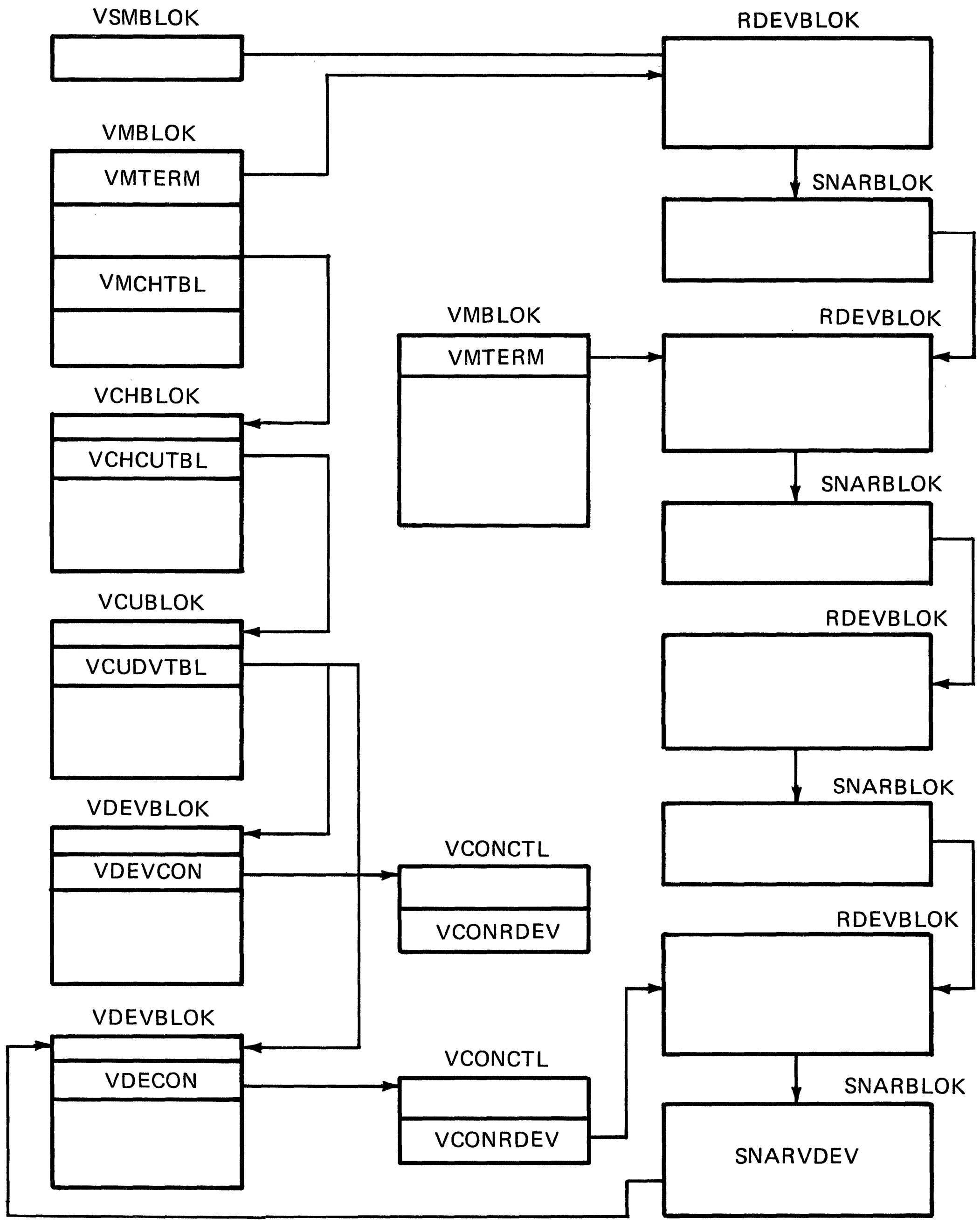


Figure 19. SNA DIAL Control Block Structure

Chapter 5. 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 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 access to MSS 3330V volume.

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

- The performance options available in VM/SP to improve the performance of a particular virtual machine
- The system options and operational characteristics of operating systems running in virtual machines that will 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 will reduce 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:

- Intercept the virtual machine SIO interrupt
- Allocate real storage space to hold the real CCW list to be created
- Translate the virtual device addresses referred to in the virtual CCWs to real addresses
- 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
- Generate a new CCW sequence building a Channel Indirect Data Address if the real storage locations cross page boundaries
- Schedule the I/O request
- Present the SIO condition code to the virtual machine
- 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 (of 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:

- 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.
- The virtual machine assist reduces the real supervisor state time used by VM/SP. See *VM/SP Planning Guide and Reference* for a list of the processors on which it is available.
- VM/370 Extended Control-Program Support further reduces the real supervisor state time used by VM/SP. See *VM/SP Planning Guide and Reference* for a list of the processors on which it is available.

Assignment and use of these options is discussed in "Virtual Machine Performance Options" on page 47.

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 user 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 will all eventually be selected 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 virtual storage may have been paged out before the time the virtual machine must begin processing. Some pages will 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 if the user re-IPLs their system by device address and specifies the CLEAR option, the locked pages are unlocked and available for use by the system being loaded. But, if the user re-IPLs their system by device address and does not specify the CLEAR option, all locked pages remain locked, with one exception - the page given to DMKVMI for IPL. Finally, if the user re-IPLs their system by name (shared system), the locked pages are only unlocked if the locked pages are not in the shared segment or if the page is in the shared segment and the user who is re-IPLing is the last user of that shared segment. In addition, issuing DIAGNOSE 14, 30, 34, or 38 against a locked page causes the page to become unlocked. The SYSTEM CLEAR command, when invoked, clears virtual machine storage and unlocks 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 will receive 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 will not be 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 "Virtual Machine Performance Options" on page 47.

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 "Virtual Machine Performance Options" on page 47.

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.

Virtual Machine Performance Options

VM/SP provides functions that create a special virtual machine environment:

1. Favored execution
2. User priority
3. Reserved page frames
4. Virtual=real option
5. Affinity
6. Multiple shadow table support
7. Shadow table bypass
8. Single processor mode
9. Dynamic SCP transition to or from native mode
10. Virtual machine assist
11. Extended Control-Program Support: VM/370.

The first nine functions are designed to improve the performance of a selected virtual machine; the last two functions improve the performance of VM/SP. Although each of the first seven functions could be applied to different virtual machines, usually they are applied to only one if optimum performance is required for that one specific virtual machine. The tenth and eleventh functions can be applied to as many virtual machines as desired.

Favored Execution

The favored execution options allow an installation to modify the normal CP deadline priority calculations in the 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 non-executable. 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.

If the favored task is highly compute bound and must compete for the processor with many other tasks of the same type, an installation can 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 of the total processor time if it can use it.[1] The favored execution option can only be invoked by a system operator with command privilege class A. The format of the command is as follows:

$$\textbf{SET FAVORED} \quad \textit{userid} \begin{bmatrix} \textit{nnn} \\ \textbf{OFF} \end{bmatrix}$$

where:

userid
: identifies the virtual machine to receive favored execution status.

nnn
: is any value from 1 through 100 and specifies the percentage of the in-queue time slice that the system will try to provide to the virtual machine.

OFF
: specifies that the virtual machine is to be removed from favored execution status

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:

```
current time-of-day +
    length of allowed processor in-queue time slice
    -----------------------------------------------
                   favored percentage
```

For example, if the processor in-queue time slice is 1 second, and the specified percentage is 10 percent (1/10), then the value added to the current

[1] The percentage of the processor time that has been requested via the SET FAVORED command with the percentage option is not an absolute value. The percentage actually received by the favored user will vary depending on the total load on the system and/or the type of load on the system. Generally it will remain relatively close to the percentage specified in the command. However, if the runlist contains multiple virtual machines which are compute bound, the favored user may not receive the requested percentage of processor time.

time-of-day is 10 seconds. The virtual machine should receive one processor time slice (1 second) once every 10 seconds.

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 prorata basis. That is, 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.

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

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:

userid
is the user's identification

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 Option

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.

Systems that are generated with the virtual=real option use the system loader (DMKLD00E). For information about generating a virtual=real system, see the *VM/SP Planning Guide and Reference.*

| Figure 20 on page 51 is an example of a real storage layout with the virtual=real option. The V=R area is 128K and real storage is 768K.

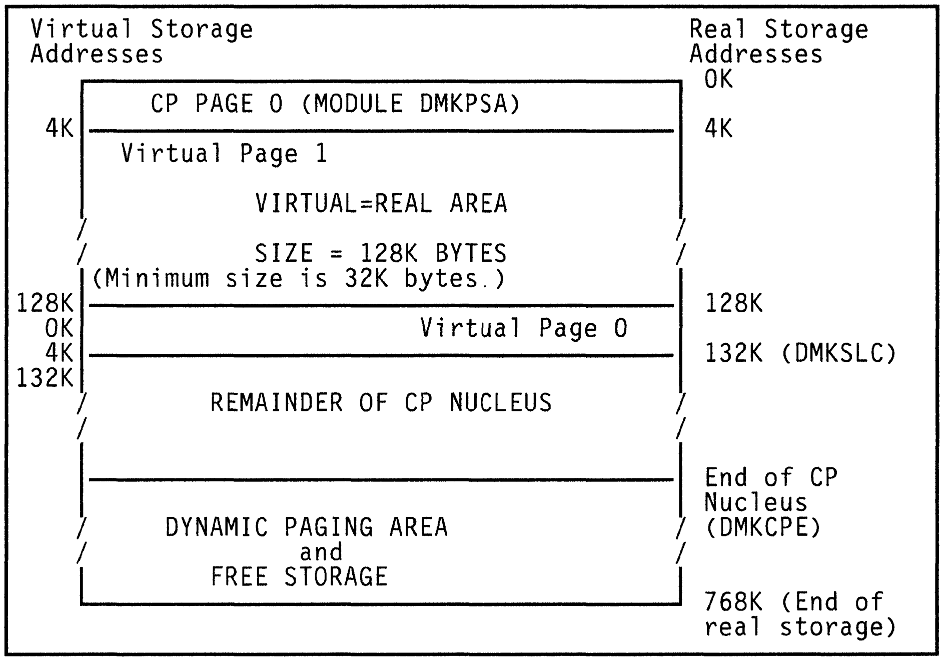

Virtual Storage Addresses
Real Storage Addresses
0K
CP PAGE 0 (MODULE DMKPSA)
4K
4K
Virtual Page 1
VIRTUAL=REAL AREA
SIZE = 128K BYTES
(Minimum size is 32K bytes.)
128K
128K
0K
Virtual Page 0
4K
132K (DMKSLC)
132K
REMAINDER OF CP NUCLEUS
End of CP Nucleus (DMKCPE)
DYNAMIC PAGING AREA and FREE STORAGE
768K (End of real storage)

Figure 20. Storage Layout in a Virtual=Real Machine

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

- 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.

- 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.

- 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)
  - Pending device status

- I/O device has an alternate path.

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

- 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.

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

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

- The reliability and availability of an operating system running virtual=real on a 3081 processor can be enhanced when the TEST BLOCK instruction (TB) is used to validate the V=R storage area.

Affinity

This virtual machine option allows virtual machines that operate on attached processor systems or multiprocessor systems to select, if desired, the processor of their choice for program execution. The selection can be made by the VM/SP directory OPTION statement, or it can be made dynamically by an operand of the CP SET command:

For class G users

SET AFFINITY $\left\{\begin{array}{l}\textit{nn}\\ \textbf{OFF}\end{array}\right\}$

For class A users

SET AFFINITY *userid* $\left\{\begin{array}{l}\textit{nn}\\ \textbf{ON}\\ \textbf{OFF}\end{array}\right\}$

where:

userid
: is the user's identification

nn
: is the processor address of a processor in an attached or multiprocessor configuration

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.

In multiprocessor configurations, both processors have the capability of executing real I/O. If the path to a user's primary minidisk or to a user's dedicated volumes are configured asymmetrically to one processor, performance benefits for the virtual machine may be derived by setting the virtual machine'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 shadow table purges when the virtual machine changes control register 1 (CR1) values, CP 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), CP can use the proper shadow table.

Multiple shadow table support adds one control block to CP, 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 last 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:

- If the virtual machine loads a new CR1 value, DMKVATAB searches the queue of STOBLOKs for the virtual CR1 value.
- If the proper STO is found, the STOBLOK is ordered first on the queue.
- If the proper STO is not found, the maximum STO count is checked.

- If the number of STOBLOKs equals the maximum STO count, DMKVATAB steals the last STOBLOK, purges the shadow tables, and reinitializes and reuses the STOBLOK by chaining it first on the queue.

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

Multiple shadow table support is controlled by the SET STMULTI command.

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 command.

Shadow Table Bypass for the V=V User

This technique is based on several characteristics of VS systems:

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

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

- 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 PTLB instruction before re-dispatching the VS system's virtual machine.

In addition, whenever a shadow table is purged because a page frame was stolen from above the high-water mark or because the virtual machine executed a PTLB or LCTL instruction, 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 with control register 1 pointing to its own virtual page and segment tables.

Single Processor Mode

In single processor mode, VM/SP uses one processor in a multiprocessing (MP) or attached processing (AP) system, while the V = R virtual machine has the exclusive use of the other processor.

VM/SP runs in uniprocessor mode (UP) with prefixing, and leaves absolute page zero for the use of the V = R virtual machine. MP-type privileged instructions issued by the V = R virtual machine are simulated by DMKPRV and DMKFPS. External interruptions are reflected to the V = R virtual machine by DMKPSA.

This function is invoked by the CP command SPMODE ON if the current VM/SP system is running in UP mode. The system must have the multiprocessor feature installed on the hardware and a V = R area must exist. If these conditions are not met, an error message is issued.

The CP command SPMODE is used to reset VM/SP to the normal UP mode of operation without prefixing. If the V = R virtual machine is still performing the prefixing function, an error message is issued.

Virtual prefix simulation is done by modifying the VMSEG segment tables and the shadow segment tables. The V = R virtual machine's prefix value is stored in segment zero of the page table entry for virtual page zero.

The VM/SP prefix value is stored in the segment's page table entry for the V = R virtual machine's prefix page. Thus, when not in virtual translate mode (a virtual machine running with an EC mode PSW with the translate bit on), access to virtual page zero is through the V = R virtual machine's prefix value that is already stored in its page table. If the virtual machine accesses its virtual prefix page, absolute page zero is accessed through reverse prefixing.

When the V = R virtual machine is in translate mode, prefix simulation is accomplished through a shadow segment table that is essentially a copy of the virtual machine segment table. A private shadow page table is maintained for segment zero and for the segment containing the virtual prefix page. The shadow page table entries for virtual page zero and the virtual prefix page are maintained in the same way as they are maintained in the VMSEG segment tables.

Since shadow tables are necessary to simulate virtual prefixing, the SET STBYPASS VR command is effectively treated as a no-operation function. The SET STMULTI command should be used to minimize shadow table maintenance overhead. In addition, the V=R virtual machine running in single processor mode can issue the SET STBYPASS command to set a high-water mark and does not require the STFIRST directory option.

When a PSW RESTART function is invoked under normal use, it forces VM/SP to take a storage dump (PSA002). When the V=R virtual machine is running in single processor mode, invoking the PSW RESTART function causes a restart interruption to be reflected to the V=R virtual machine.

If a forced VM/SP storage dump (PSA002) is required when running in single processor mode, a flag byte in DMKPSA must be set before invoking the PSW RESTART function.

Information about the commands used in this support is found in the *VM/SP CP Command Reference*. A full description of the operating procedures for this support is contained in the *VM/SP Running Guest Operating Systems*.

Dynamic SCP Transition to or From Native Mode

This function allows the operating system running in the V=R virtual machine to make a transition from the VM/SP environment to the native environment and back again to the VM/SP environment. It eliminates the necessity of system shutdown and reinitialization for the operating system and VM/SP.

To make the transition to native mode, the indicated users must perform the following steps:

- All I/O used by the V=R virtual machine must be dedicated and the virtual I/O addresses must be the same as the real I/O addresses.
- The virtual machine operator must stop the virtual spooling devices (non-dedicated) and detach them from the virtual machine.
- The VM/SP operator must drain the real unit-record devices.
- All users, except the V=R virtual machine and the VM/SP system operator, must be logged off.

When these conditions are met the VM/SP system operator can issue the CP command:

QVM *userid*

The VM/SP machine's page zero is saved and the V=R virtual machine's page zero is moved to absolute page zero. The timers, control registers, and general registers are initialized to the virtual machine's values. Control is then given to the V=R virtual machine in native mode.

The operating system in the V=R virtual machine is not given native control if any of the following conditions are detected:

- CP wait conditions are present.
- I/O interruptions are pending.
- External interruptions are pending.
- I/O requests are queued within VM/SP.
- An ADSTOP has been set or tracing is active.

The transition to native mode is handled by the DMKQVMRT subroutine in the DMKQVM module.

This support also allows the transition from native mode back to VM/SP. When the V=R virtual machine is given native control, the RESTART PSW is modified to point to an entry point in the DMKQVM VM/SP module.

The system operator must set a flag byte within DMKQVM and perform a PSW RESTART function. This gives control to a subroutine in the DMKQVM VM/SP module (DMKQVMRS for non-370E operating systems, DMKQVMRX for 370E operating systems), and the transition from native mode back to the VM/SP environment is performed. After this transition is completed, the operating system is running in the V=R virtual machine again.

When making the transition back to the VM/SP environment, the operating system operator should ensure that the operating system's I/O configuration and environment are the same as when the transition to native mode took place. If a PSW RESTART function is invoked while running in native mode and without setting the flag byte to return to VM/SP mode, a restart interruption is reflected to the native system.

Information about the commands used in this support is found in the *VM/SP CP Command Reference*. A full description of the operating procedures for this support is contained in the *VM Running Guest Operating Systems*.

Virtual Machine Assist Feature

The virtual machine assist feature is a processor hardware feature. It 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; and, consequently, VM/SP performance is improved. See the Functional Characteristics Document for your processor model to see if virtual machine assist is available.

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.

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.

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. The format of the system operator's SET command is:

```
SET SASSIST [ON ]  [[PROC] xx]
            [OFF]
```

where:

PROC *xx*
is the address of the processor (0-63).

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 CP Command Reference.*

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. 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. The format of the command is:

```
SET ASSIST {[ON]  [SVC  ]  [TMR  ]}
           {      [NOSVC]  [NOTMR]}
           {OFF                   }
```

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.*

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 if it:

- Has an instruction address stop set (ADSTOP).
- Traces 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 will indicate 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 will indicate the assist feature is off.

Extended Control-Program Support (ECPS) for VM/370

Extended Control-Program Support for VM/370 (ECPS:VM/370) improves the performance of the processor when executing VM/SP beyond the improvement attained by the virtual machine assist feature described above. See *VM/SP Planning Guide and Reference* for a list of the processors on which ECPS:VM/370 is available. ECPS:VM/370 consists of three parts:

- CP Assist - The CP assist part of ECPS assists various CP routines that are frequently used by the system. Because these routines are assisted by the hardware without involving CP, performance is improved.

- Expanded Virtual Machine Assist - The expanded virtual machine assist extends the level of handling of certain privileged instructions. It also processes certain privileged instructions not handled by the virtual machine assist feature.

- Virtual Interval Timer Assist - The virtual interval timer assist[2] provides hardware updating of the virtual interval timer at virtual location X'50'. This results in an update frequency of approximately 300 times per second, the same as for the real interval timer. Procedures that use the virtual interval timer for job accounting, performance measurements, and the like, will therefore generate more accurate and repeatable time data than they would if the virtual timer was being updated by CP routines. Timer updating occurs only while the virtual machine is in control of the real processor.

See Appendix A, "Extended Control-Program Support" on page 401 for additional information.

Using the Extended Control-Program Support:VM/370

Extended Control-Program Support (ECPS) is controlled at two levels:

- VM/SP system
- Virtual machine.

[2] These functions and instructions comprise the machine implemented package that is available on the 3031 uniprocessor and 3031 attached processor.

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

SET CPASSIST OFF

will disable 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

will enable 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.

Virtual Machine Communication Facility

The Virtual Machine Communication Facility (VMCF) allows any logged-on user of VM/SP to transfer messages, control data, data files, or combination of all three to another virtual machine running under the same VM/SP system. Information is transferred directly from one virtual storage to the other virtual storage with CP buffering the information. Only one data page frame must be locked at any one time. The amount of data that can be transferred is limited only by the virtual storage sizes of the virtual machines involved.

VMCF contains five data movement and seven control functions and is invoked by a virtual machine via the DIAGNOSE interface (code X'0068'). A special external interrupt code, X'4001', notifies a virtual machine that a VMCF communication is pending. A virtual machine can have a maximum of 50 messages active at any one time. The number of messages is an equate in the DMKVMC module and can be changed to accommodate different VM/SP storage sizes.

VMCF Diagnose Interface

When a virtual machine issues a DIAGNOSE instruction with a function code of X'0068', the Rx register contains the virtual address, doubleword-aligned, of a 40-byte parameter list. This parameter list (VMCPARM) contains a hexadecimal code to identify the specific VMCF subfunction. It also contains the data addresses, data lengths, and control information that are required to execute the particular subfunction.

The DIAGNOSE instruction, a privileged operation, is processed by DMKPRV which passes control to DMKHVC, the DIAGNOSE interface module. DMKHVC, in turn, validates the function code and, if the code is X'0068', turns control over to DMKVMC, the VMCF module. DMKVMC validates the VMCPARM address and length, the subfunction code, and passes control to the appropriate subroutine. The VMCF subfunctions and their codes are as follows:

Code	Subfunction
X'0000'	Allow virtual machine communication
X'0001'	Disallow virtual machine communication
X'0002'	Initiate a SEND request
X'0003'	Initiate a SEND/RECV request
X'0004'	Initiate a SENDX request
X'0005'	Accept data from a SEND or SEND/RECV request
X'0006'	Cancel specific request you initiated
X'0007'	Reply to a SEND/RECV request
X'0008'	Quiesce incoming communications
X'0009'	Resume accepting communications
X'000A'	Notify a user that you are ready for communications
X'000B'	Reject a specific incoming communication

Special VMCF External Interrupt

Whenever a source virtual machine uses VMCF to correspond with another virtual machine (sink), the sink is notified of the pending communication via a special external interrupt (code X'4001'). When this interrupt is unstacked and processed, a copy of the information in the source's parameter list is passed to the sink in an external interrupt buffer. The buffer is defined when a user allows virtual machine communication. The contents are referred to as the external interrupt message header.

When certain transactions (SEND, SEND/RECV, SENDX) have been completed, a final response external interrupt is passed back to the source. The message header associated with this interrupt contains residual counts pertaining to the transferred data and data transfer return codes.

VMCF Control Blocks and Data Areas

Figure 21 on page 63 shows the relationship between the various VMCF control blocks and data areas. When a virtual machine allows virtual machine communication, VMCF generates a master VMCBLOK and places it at the head of a queue pointed to by the VMCPNT field of the user's VMBLOK. Two fields in this master VMCBLOK define the address (VMCVADA) and length (VMCLENA) of the user's external interrupt buffer. The length must include the maximum size of any potential SENDX data in addition to the 40 bytes for the external interrupt message header.

When a source virtual machine executes a VMCF subfunction, a VMCBLOK is built, initialized with data from the parameter list (VMCPARM), and stacked on the VMCBLOK queue pointed to by the VMCPNT field in the sink's VMBLOK. If an XINTBLOK for a X'4001' external interrupt has not already been stacked for the sink machine, DMKVMC builds one and stacks it on the XINTBLOK queue pointed to by the VMPXINT field in the sink's VMBLOK. VMCF external interrupts are assigned a sort code of X'7FFFFFFF', giving them the lowest priority in the external interrupt queue. Each virtual machine clears its own VMCF control blocks.

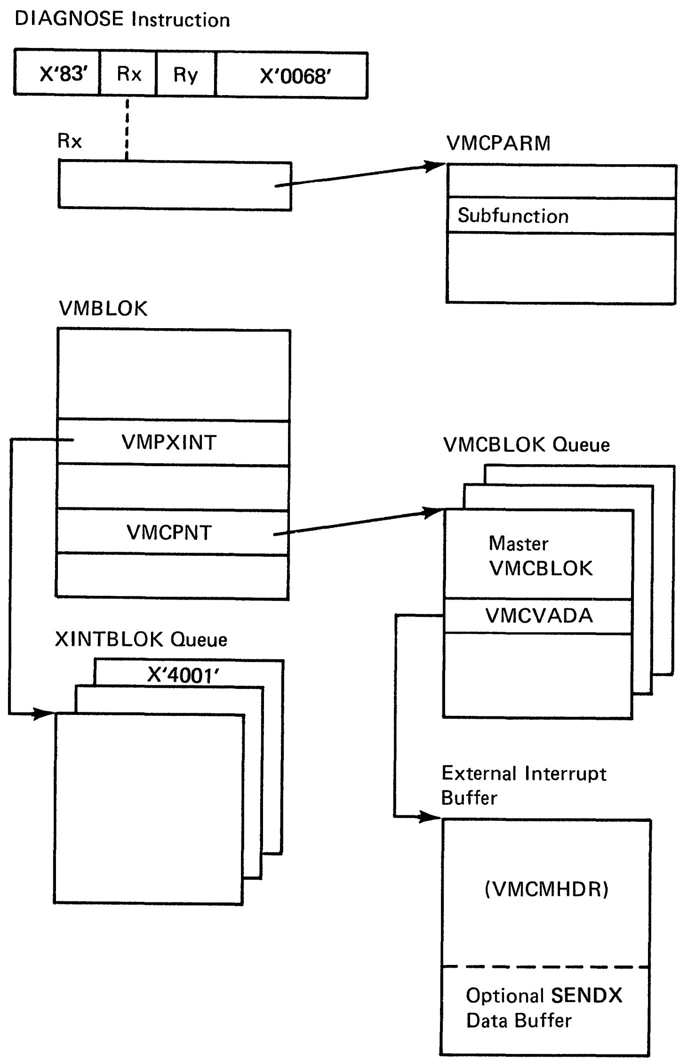


Figure 21. VMCF Control Block Relationships

Special Messages Facility

The Special Message Facility allows a user on one virtual machine to send special messages to another virtual machine via the SMSG command. The Special Messages Facility may be used with the Virtual Machine Communications Facility (VMCF) or the Inter-User Communications Facility (IUCV) to send the messages. In the Special Message environment, CP acts as a source machine with the receiver of the special message being the sink. This relieves the burden from the issuer of SMSG of having to perform authorization and other setup necessary for sending messages to the receiving virtual machine. Authorization is performed by CP.

The issuer of SMSG is responsible for sending message text that is meaningful to the receiving virtual machine. The format and handling of special messages is entirely up to the receiving machine, which may be one designed by the installation or prepared by others.

Before the receiving virtual machine can accept special messages, it must be running with the Special Message flag ON, and it must have issued AUTHORIZE (via DIAGNOSE X'68') with CP. The authorization includes supplying the External Interrupt Buffer address and size. To ensure receiving the entire message, the receiving virtual machine should specify the size as 280 bytes (room for a 40-byte header and a 240-byte message buffer).

Note: 'MSG TOO LARGE' condition may occur if the 'SMSG' command is issued on a 3279 or 3278 mod 5 terminal to send a message longer than what the receiving virtual machine has specified.

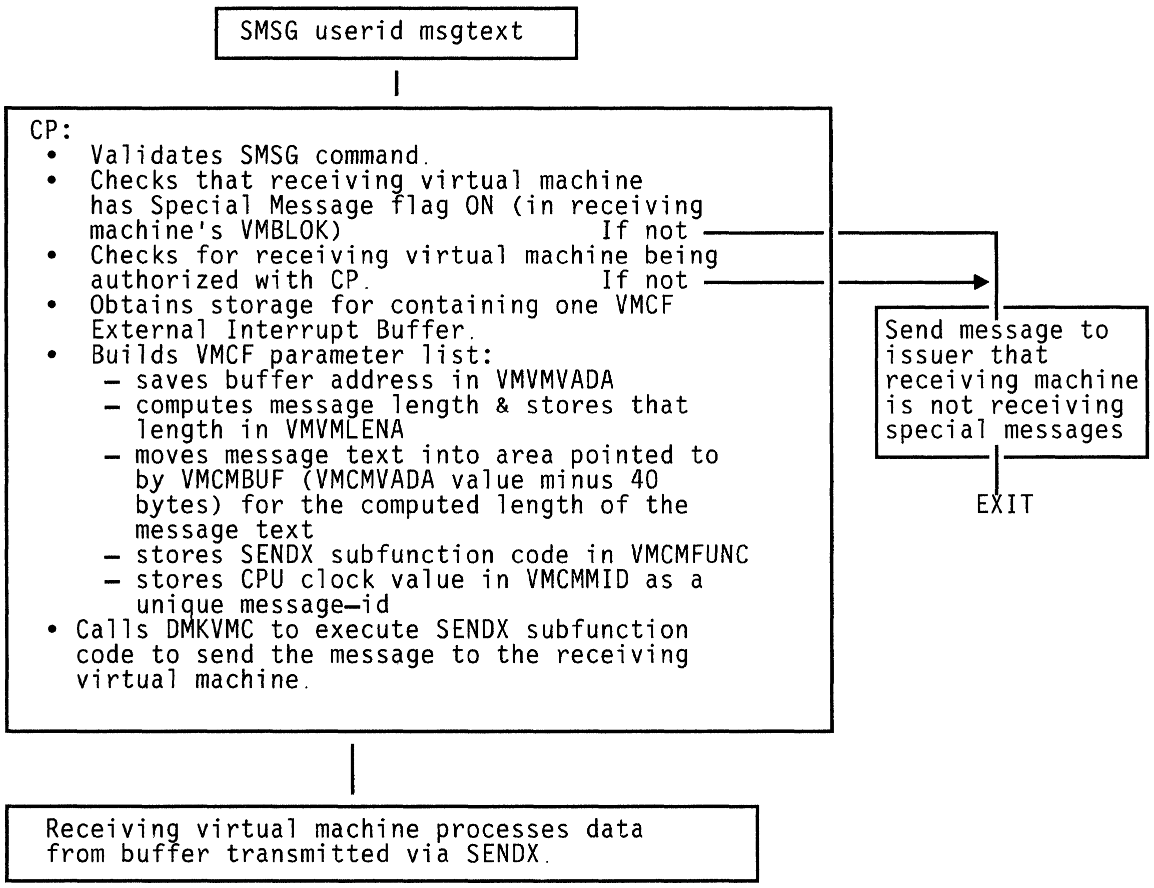
Setting SMSG ON can be accomplished by setting the SMSG flag on in the VMCF parameter list when issuing an AUTHORIZE. It may also issue the CP command SET SMSG ON. Either method sets the Special Message flag on in the VMBLOK. When this is done, any other virtual machine can issue the SMSG command to the userid of the receiving virtual machine.

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.

If the receiving virtual machine chooses not to accept special messages at any time, it can merely issue SET SMSG OFF. CP would then inform any machine issuing the SMSG command that the virtual machine is not receiving special messages. When it is ready to resume accepting special messages, the virtual machine need only to issue SET SMSG ON or SET SMSG IUCV.

Figure 22 on page 65 shows the processing when an SMSG command is issued.



```
SMSG userid msgtext

CP:
 •  Validates SMSG command.
 •  Checks that receiving virtual machine
    has Special Message flag ON (in receiving
    machine's VMBLOK)                  If not
 •  Checks for receiving virtual machine being
    authorized with CP.                If not
 •  Obtains storage for containing one VMCF
    External Interrupt Buffer.
 •  Builds VMCF parameter list:
    – saves buffer address in VMVMVADA
    – computes message length & stores that
      length in VMVMLENA
    – moves message text into area pointed to
      by VMCMBUF (VMCMVADA value minus 40
      bytes) for the computed length of the
      message text
    – stores SENDX subfunction code in VMCMFUNC
    – stores CPU clock value in VMCMMID as a
      unique message–id
 • Calls DMKVMC to execute SENDX subfunction
   code to send the message to the receiving
   virtual machine.

Send message to
issuer that
receiving machine
is not receiving
special messages

EXIT

Receiving virtual machine processes data
from buffer transmitted via SENDX.
```

Figure 22. SMSG Command Processing

Inter-User Communications Vehicle

There are 18 IUCV functions with which you can initiate, interrogate, receive, reply to, and terminate individual communications. You can interrogate, receive, or terminate any communication over a particular path. You can interrogate and/or terminate any communication over any path. The IUCV functions are:

ACCEPT	QUIESCE	SEND
CONNECT	RECEIVE	SET CONTROL MASK
DECLARE BUFFER	REJECT	SET MASK
DESCRIBE	REPLY	SEVER
PURGE	RESUME	TEST COMPLETION
QUERY	RETRIEVE BUFFER	TEST MESSAGE

ACCEPT Function

The target communicator uses the ACCEPT function to respond to a connect request from a virtual machine. If the target communicator chooses not to complete the connection, it invokes the SEVER function. If the ACCEPT function initiator does not declare a buffer, the system generates a specification exception.

The ACCEPT function initiator receives a return code if one of the following occurs:

- The path specified is not a pending connection.
- The path has been severed by the originator of the CONNECT function.
- An IUCV ACCEPT was issued in response to an APPC/VM CONNECT or an APPC/VM ACCEPT was issued in response to an IUCV CONNECT.

Both the initiator and the originator have their path descriptor entries set to valid and an external interrupt is built. A request for QUIESCE mode when the originator invokes CONNECT results in an indicator being set in both the path descriptor and the external interrupt. The QUIESCE option is ignored for APPC/VM CONNECTs and ACCEPTs.

When the ACCEPT function is invoked from CP system code, the message limit and priority setup is based on the parameter list values. If the message limit is not specified, a default value is used. When an APPC/VM ACCEPT is invoked, the message limit for the path is one.

When the ACCEPT function is not invoked from CP system code, a directory check is made to determine whether or not the target communicator entry exists. The directory check search sequence is:

- The initiator's IUCV entries to find an entry for the source communicator
- The initiator IUCV entries to find an entry for ALL.

If neither of these authorizations exists, then communications are not allowed.

For the message limit, the lower value of the parameter list limit or the directory limit is used. If you specify neither, the default value prevails. The established message limit returns to the initiator in the parameter list and then passes to the source communicator in the connection complete external interrupt.

If the communicator wishes to receive data in the parameter list as well as in a buffer, the communicator can use the PARMDATA=YES option. Parameter list data is limited to two fullwords, or eight bytes, of data. However, parameter list data is not permitted on APPC/VM paths. The PARMDATA=YES option is ignored for APPC/VM ACCEPTs.

If the originator of the IUCV CONNECT or APPC/VM CONNECT is a virtual machine, and WAIT=NO was specified, an external interrupt is stacked and the initiator receives a normal return code. If the originator of the APPC/VM CONNECT specified WAIT=YES, then the CPEXBLOK created during CONNECT processing is stacked to complete the originator's CONNECT and the initiator receives a normal return code.

If the originator is CP system code, the pending connection is located and dequeued from the CP pending connection chain and a CPEXBLOK is stacked to indicate to the originator that the connection is complete. Finally, the initiator receives a normal return code.

CONNECT Function

To start communication between virtual machines or between a virtual machine and a CP system, use the CONNECT function. During the CONNECT process:

- If a buffer is not declared, a specification exception is generated.
- If the source communicator is a CP system service, directory checking is skipped.
- If the target communicator is not logged on or has not declared a buffer, a return code is sent to the initiator.
- If the CONNECT function is invoked from CP system code, the message limit and priority setup is based on the parameter list values. If the message limit is not specified, a default value is used.
- If the target communicator is a CP system service (where the ID starts with an asterisk) the name is verified by looking it up in a table. If not found, a return code is sent to the initiator.
- If directory checking is needed, the directory is checked to verify that this connection is authorized. The directory check search sequence is:
  - The initiator's IUCV entries to find an entry for the target communicator
  - The initiator's IUCV entries to find an entry for ALL.
- When directory authorization is not found, a return code is sent to the initiator.
- If the communicator wishes to receive data in the parameter list as well as in a buffer, the communicator can use the PARMDATA=YES option. Parameter list data is limited to two fullwords, or eight bytes, of data. However, parameter list data is not permitted on APPC/VM paths. The PARMDATA=YES option is ignored for APPC/VM CONNECTs.
- When an APPC/VM CONNECT is invoked, the message limit for the path is one.

The message limit is specified either in the IUCV directory control statement, which provides connection authorization, or in the parameter list, or in both. When a message limit is specified in both, use the lower one of the two. The established message limit returns to the initiator in the parameter list and then passes to the target communicator in the pending connection external interrupt.

In the directory entry, priority authorization is allowed only if specified, for IUCV CONNECTs. For APPC/VM CONNECTs, priority is ignored.

If either the initiator or the target communicator exceeds its maximum connection limit, the initiator receives a return code.

When the target communicator is a virtual machine, a pending connection external interrupt is stacked for it. When the target communicator is a CP system service, the connect entry point for that service is invoked. When the originator is CP system code, a pending connection is added to the CP pending connection chain. When APPC/VM CONNECT with WAIT=YES is specified, a CPEXBLOK is obtained and chained off the initiator's IUCVBLOK. IUCV then exits to the dispatcher to wait for the CPEXBLOK to be stacked. After the CPEXBLOK is stacked, the initiator receives an indication that the CONNECT is complete.

DECLARE BUFFER Function

To specify the buffer address where the virtual machine external interrupt should be stored, use the DECLARE BUFFER function.

It is unnecessary for CP system code to issue the DECLARE BUFFER function, because CP does not receive external interrupts.

If the initiator already has a buffer, the initiator receives a return code.

Based on the maximum number of connections, the system builds an IUCVBLOK for the initiator. The directory contains the maximum number of connections permitted for each user; if none appears, the default is four. When errors are encountered reading the directory, the initiator receives a return code.

Note: 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.

Note: The IUCV external interrupt submask bits are set on when executing the DECLARE BUFFER function. You must invoke the SET MASK function to change any of these bit settings.

DESCRIBE Function

The target communicator uses the DESCRIBE function to determine the presence of any messages not previously described or reflected in a message pending IUCV external interrupt. The parameter list will receive the pertinent information about the message.

Search of the SEND queue uses the first-in-first-out (FIFO) method. The first undescribed and unreflected MSGBLOK found is selected. APPC/VM MSGBLOKs are only described if the corresponding path is in RECEIVE state. Because of their position in the SEND queue, priority MSGBLOKs are scanned first. The DESCRIBE function causes the following MSGBLOK information to be stored:

- Path ID
- Target message class
- Message ID
- Message flags
- Length of message
- Length of answer area.

Note: Target message class, message ID, and length of answer area are not stored for APPC/VM messages.

This information eventually allows the target communicator to accept the MSGBLOK data through use of the RECEIVE function.

The DESCRIBE function sets the condition code to indicate:

- No undescribed MSGBLOK found
- MSGBLOK found; description stored.

If the message being described contains the message data in the parameter list, or is an APPC/VM SENDREQ MSGBLOK, the message is considered to be received and the MSGBLOK is removed from the SEND queue.

Note that IUCV describes each MSGBLOK once and the target communicator is responsible for removing those MSGBLOK(s) from the SEND queue. Use of either the RECEIVE function or REJECT function removes the MSGBLOK(s).

The DESCRIBE function clears the pending message external interrupt for each MSGBLOK.

Note: CP system code (aside from IUCV support) cannot use the DESCRIBE function.

PURGE Function

The source communicator uses the PURGE function to terminate a single message. The message is destroyed immediately under the following circumstances:

- If it has not been described to the target communicator, which means the target communicator is completely unaware that the message was ever sent
- If it is on the source communicator's REPLY queue.

However, the message, when previously described to the target communicator, is marked as purged and the target communicator receives a return code in the parameter list when next invoking either the RECEIVE function or the REPLY function, whereupon the message is destroyed. Note that the PURGE function moves no data.

The PURGE function parameter list describes the to-be-purged message; this list includes:

- A path ID (The value of the path ID determines which target communicator queues are to be searched.)
- A message ID.

When both of these are specified, then the source message class must also be specified. However, when the message ID is not specified, then the message class is optional.

Note: The parameter list flags indicate which fields are to be used in locating the message.

The search for the message starts with the source communicator's REPLY queue, then the target communicator's RECEIVE queue, and then the target communicator's SEND queue. The first MSGBLOK that matches the message specification terminates the search. When a message is purged, the message ID, the path ID, the source message class, the message tag, the message flags, and the audit trail are all stored in the parameter list.

A condition code reflects the completion status of the PURGE function. Detection of certain error conditions stores a return code, which indicates the type of error found.

QUERY Function

Use the QUERY function to extract IUCV information about the virtual machine. The IUCV external interrupt buffer size is returned in general register 0. General register 1 holds the data for the maximum number of connections outstanding for the particular virtual machine.

Notes:

1. *The QUERY function does not take a parameter list.*
2. *If errors are encountered, a condition code is set.*
3. *CP system code cannot invoke the QUERY function.*

QUIESCE Function

Use the QUIESCE function when you wish to suspend temporarily the ability of the communicating partner to send messages.

When the request is to quiesce all paths, each path quiesces individually and the initiator receives a return code. For each path on a QUIESCE ALL or for the single specified path, (if the path is marked severed or if the path is already quiesced) the initiator receives a normal return code. Otherwise, the system builds an external interrupt based on the path being quiesced, and marks the path as quiesced.

When the communicating partner is a CP system service, the quiesce entry point for the service is located and external interrupt data is passed to it via a CALL linkage. The initiator receives a return code upon return of control.

If a virtual machine is the communicating partner, the external interrupt for the communicating partner is stacked and the initiator receives a return code.

RECEIVE Function

The target communicator uses the RECEIVE function to accept messages.

When the complete message moves from the send area to the specified receive area, the MSGBLOK for the designated message moves from the SEND queue to the RECEIVE queue. If the receive area cannot contain the message, the MSGBLOK stays in the SEND queue and the length of the remaining data is stored in the parameter list. When the RECEIVE function is again activated, the remainder of the message becomes available.

RECEIVE function input specifies which message is to processed; identified by:

- Message ID
- Path ID
- Target message class.

Note: Message ID and target message class are not valid for APPC/VM RECEIVEs.

If the message ID is not specified, then any combination of path ID and target message class can be specified. The flag fields of the RECEIVE parameter list indicates the type of message description to be used. If no parameter list search fields are specified, for IUCV RECEIVEs, the first message that has not been partially received is presented.

If message data is contained in the parameter list and a REPLY is expected, the MSGBLOK is moved from the SEND queue to the RECEIVE queue. On a one-way (no REPLY) message, the MSGBLOK is destroyed.

Note: After a partially received message, the message must be completely specified in order to receive the remainder of the message.

The target communicator obtains the required message description in one of the following ways:

- By use of the DESCRIBE function
- Via presentation by an external interrupt.

Note: A different linkage is used for a CP system service because neither the DESCRIBE function nor an external interrupt is available.

The RECEIVE function input identifies a receive area to which the message goes if BUFLIST = YES is not specified. A beginning address and a length describe the receive area. The address must be real to the virtual machine, and no alignment is required for the beginning address. Data movement terminates when the message length exceeds the receive area length or the message ends.

Note: If length specification is 0 for either data area, there is no data movement.

If BUFLIST = YES is specified, the BUFFER = parameter of the IUCV macro instruction provides the address of a list of addresses and lengths of discontiguous buffers containing the message text. This list of addresses must be on a doubleword boundary; however, addresses within the list need not be doubleword-aligned. Also, the value specified with the BUFLEN = parameter is the total of the individual buffer lengths in the list pointed to by BUFFER = .

Completion of the RECEIVE function results in update of the parameter list receive area. If BUFLIST = NO, the address is set to the originally designated length plus the number of bytes moved. The updated length is the residual count when the return code indicates that the buffer was too short, or the return code indicates that it is the remaining length of the buffer.

If BUFLIST = YES is specified, the address in the parameter list points to the entry in the buffer list to continue processing. The buffer list is updated throughout IUCV processing. As data is moved, the address in the list entry is incremented by the length moved, and the length in the list entry is decremented by that length. When the length is zero, the list pointer is incremented to indicate the next entry to process. The total length specified in the parameter list is also decremented with each move. It reflects the amount of data yet to be received.

The MSGBLOK moves directly to the REPLY queue when either the send area or the receive area have addressing exceptions and/or protection exceptions. When this happens, the target communicator receives a return code to that effect while the audit trail notifies the source communicator.

A condition code is set to report RECEIVE function completion status. If an error condition occurs, its detection activates the setting of a return code to indicate the type of error.

When a priority MSGBLOK moves to the RECEIVE queue, the concept of priority disappears. The REPLY function can reintroduce priority.

REJECT Function

To reject a single message from the source communicator, the target communicator executes the REJECT function. The MSGBLOK for the designated message moves from the target communicator's SEND queue or RECEIVE queue to the source communicator's REPLY queue. Use of the REJECT function moves no data.

REJECT function input specifies which message is to be rejected. The information is identified by:

- Message ID
- Path ID
- Target message class.

If the message ID is not specified, then any path ID and target message class combination are valid. The flag field of the REJECT parameter list indicates the type of message description to be used. If flags are not specified, the REJECT function is terminated with a specification exception.

The program searches the target communicator's queues for the designated message: first, the RECEIVE queue; then, the SEND queue. The first MSGBLOK that matches the designated message moves to the source communicator's REPLY queue. Rejection of the designated message is via setting a condition code.

The REJECT function parameter list stores the message ID, the path ID, and the target message class on completion.

To indicate message rejection, the audit trail is updated.

A condition code is set to report REJECT function completion status. If certain types of errors occur, storing of a return code indicates it.

REPLY Function

The target communicator uses the REPLY function to respond to a message. The MSGBLOK for the designated message moves from the target communicator's RECEIVE queue to the source communicator's REPLY queue. Data moves from the specified reply area to the source communicator's answer area.

It is assumed that REPLY function input has a complete description of the designated message requiring a reply. Partial descriptions of a message are not supported. If insufficient information is supplied the parameter list, locating the designated message will be impossible. The message description comprises:

- Message ID
- Path ID
- Target message class.

If the designated message is not found, a parameter list return code is set.

The target communicator turns on the parameter list flag to specify that the reply message has priority. The reply MSGBLOK thus precedes any nonpriority MSGBLOK(s) in the queue, immediately following any earlier-designated priority MSGBLOK(s).

Input to the REPLY function identifies a reply area if ANSLIST=YES is not specified. The description includes a beginning address and a length. The address must be real to the virtual machine although no alignment requirement is made for the beginning address. Data moves between the target communicator's reply area and the source communicator's answer area and terminates when either area length is exhausted. A length of 0 for either area prevents data transfer. On completion of the REPLY function, any length mismatches result in an error condition being posted.

If ANSLIST=YES is specified, the ANSBUF= parameter of the IUCV macro instruction provides the address of a list of addresses and lengths of discontiguous buffers containing the message reply text. This list of addresses must be on a doubleword boundary; however, addresses within the list need not be doubleword-aligned. Also, the value specified with the ANSLEN= parameter is the total of the individual buffer lengths in the list pointed to by ANSBUF=.

When addressing exceptions and protection exceptions occur while accessing either the reply area or the answer area, the MSGBLOK moves to the REPLY queue, a return code is sent to the target communicator, and the audit trail notifies the source communicator.

If ANSLIST=NO, completion of the REPLY function updates the reply area description in the parameter list and sets the address at the original length *plus* the number of bytes moved. Note that the updated length is the residual count if the return code indicates the buffer was too short or is the remaining length of the buffer on a normal return code.

If ANSLIST=YES is specified, the address in the parameter list points to the entry in the answer list to continue processing. The answer list is updated throughout IUCV processing. As data is moved, the address in the list entry is incremented by the length moved, and the length in the list entry is decremented by that length. When the length is zero, the list pointer is incremented to indicate the next entry to process. The total length specified in the parameter list is also decremented with each move. It reflects the amount of data yet to be moved.

The REPLY function cannot execute if a message is sent by a one-way SEND function; such a message never resides on the target communicator's RECEIVE queue. Thus, a `no message found` condition results.

A condition code reflects the REPLY function completion status. If an error condition occurs, its detection activates storing of a return code.

RESUME Function

Use the RESUME function to restore IUCV communications after you invoke the QUIESCE function.

If your request is to resume all paths, each path resumes individually and the initiator receives a return code.

After issuing a RESUME function for a single specified path or for ALL paths, (if the path is marked severed or is not quiesced) a normal return code is returned to the initiator.

If the communicating partner is not CP, the external interrupt is stacked for the virtual machine and the initiator receives a return code.

If the communicating partner is a CP system service, the program locates the RESUME entry point for the service and the external interrupt data passes to it via a CALL linkage.

When the CALL linkage processing completes, the initiator receives a return code.

RETRIEVE BUFFER Function

Use of the RETRIEVE BUFFER function notifies IUCV that the virtual machine no longer needs IUCV.

The program generates a SEVER ALL to terminate all messages on all paths for the designated communicator. Any control blocks built for this communicator at DECLARE BUFFER time are dismantled and released.

Note: The RETRIEVE BUFFER functions cannot be used in CP system code outside of IUCV support.

SEND Function

The SEND function initiates communication by creating a MSGBLOK and enqueuing it on the target communicator's SENDQ.

The input to the SEND function must completely describe the message being sent. It must specify the source communicator's path ID and the source communicator's and target communicator's message classes. Also required when invoked from a virtual machine is the message tag that is presented to the source communicator upon completion of the message. The MSGTAG field is used by IUCV for CP-initiated messages and is not available to a CP service.

Note: Message class and message tag are not valid on APPC/VM SENDs.

If BUFLIST = YES and ANSLIST = YES are not specified, the user can specify two data areas that are used to move data between the source communicator and target communicator. The send area contains the data to be moved from the source communicator to the target communicator. The answer area is the area into which the target communicator's REPLY data is moved, for IUCV SENDs. For APPC/VM SENDs, the target communicator's SENDDATA data is moved into the answer area. Each area is defined by a beginning address and a length. Each address must be real to the virtual machine. Either data area can be anywhere within the source address space. There is no boundary alignment requirement on the beginning addresses.

The user may choose to send the data in the parameter list and not specify a send area; or the user may choose to have the REPLY data returned in the parameter list and not specify an answer area. When using the parameter list data option, the user is limited to two fullwords, or eight bytes. Parameter list data is not valid on APPC/VM SENDs.

If BUFLIST = YES is specified, the BUFFER = parameter of the IUCV macro instruction provides the address of a list of addresses and lengths of discontiguous buffers containing the message text. This list of addresses must be on a doubleword boundary; however, addresses within the list need not be doubleword-aligned. Also, the value specified with the BUFLEN = parameter is the total of the individual buffer lengths in the list pointed to by BUFFER =.

If ANSLIST = YES is specified, the ANSBUF = parameter of the IUCV macro instruction provides the address of a list of addresses and lengths of discontiguous buffers containing the message text. This list of addresses must be on a doubleword boundary; however, addresses within the list need not be doubleword-aligned. Also, the value specified with the ANSLEN = parameter is the total of the individual buffer lengths in the list pointed to by ANSBUF =.

The IUCV SEND function does not move any data. The target communicator invokes the RECEIVE and/or REPLY functions to move data. Because of this, a description of the send area and the answer area is stored in the MSGBLOK for use during either function. The description consists of the beginning buffer or list addresses, the total length of each area, and the PSW Key to be used for protection checking during access to each area. The APPC/VM SEND function moves the data if a receive or answer area has been previously defined by the target for the path.

Data areas are not validity checked during the SEND operation, unless the data is actually moved. The check occurs when the areas are used. Access exceptions in the source address space are recognized and reported during processing of the RECEIVE function and/or REPLY function.

Using a parameter list flag field, you can optionally alter the SEND function to either a priority message or a one-way message. The SEND function with the priority flag set enqueues the MSGBLOK on the target communicator's SEND queue preceding all nonpriority MSGBLOK(s) and following all earlier priority MSGBLOK(S). The SEND function with the one-way flag set designates the MSGBLOK as one that does not allow a reply. When the target communicator receives a one-way message, the MSGBLOK skips the target communicator's RECEIVE queue, and is immediately placed on the source communicators REPLY queue.

A condition code is set to report SEND function completion status. If an error condition occurs, its detection activates the setting of a condition code as well as the storing of a return code to indicate which error was detected.

SET MASK Function

The SET MASK function enables or disables external interrupts for priority messages and nonpriority messages, priority replies and nonpriority replies, and IUCV controls. Specify all mask bits in the parameter list. All mask bits are used and override any and all previous mask specifications. Use of the mask is in addition to the global external interrupt mask in the PSW.

Note: The SET MASK function cannot be used from CP system code.

SET CONTROL MASK Function

The SETCMASK function enables or disables external interrupts for the five types of IUCV control interrupts. These interrupts include: Connection Pending, Connection Complete, Path Severed, Path Quiesced, and Path Resumed. Specify all mask bits in the parameter list. All mask bits are used and override any and all previous mask specifications.

Before the Control Mask bits are interrogated, a virtual machine must first be enabled for IUCV control type external interrupts by using the SETMASK function.

Note: The SETCMASK function cannot be used from CP system code.

SEVER Function

Use the SEVER function to terminate IUCV communications capabilities.

If the path is complete, both communicators must invoke a SEVER function to the path. After one communicator's invocation, all messages still on the path terminate and the communicating partner receives a sever external interrupt. The communicating partner then, if desired, can dequeue the terminated message(s). When finished, the communicating partner invokes the SEVER function.

For the SEVER ALL, each message on either the SEND queue or the RECEIVE queue is designated as severed in the audit trail. Then the REJECT function is invoked to terminate each message. The communication control table (CCT), part of the IUCVBLOK, contains a list of the valid path ID(s). Once message termination is complete, each valid path from 0 through the highest valid path in the CCT also terminates. The process completes with the release of all the space used by the identification control blocks. All designated paths are masked as invalid.

Path termination for a valid path proceeds as follows:

- If the path is invalid or out of range, the initiator receives a return code.

- If the path is valid, the QUIESCE function is invoked to prevent further communication.

- If the path is marked as severed, the entry is set to available and the initiator receives a return code. Any messages in the REPLY queue for the path are dequeued and the space returned to storage.

- For each message on either the SEND queue and/or the RECEIVE queue for this path, a REJECT is issued.

- Each message generated by the initiator for this path terminates with a PURGE operation.

If the communicating partner is a CP system service, the program locates the sever entry point for the service and the external interrupt data passes to it via a CALL linkage.

When the CALL linkage processing completes, the initiator receives a return code.

If the communicating partner is not CP, the external interrupt is stacked for the virtual machine and the initiator receives a return code.

Path termination for an invalid path proceeds as follows:

- If the path is severed because the SEVER function has been invoked from the target communicator, the path is set to available and the initiator receives a normal return code.

- If the communicating partner has received the pending connection interrupt, the blocks are marked as severed and the communicating partner must also invoke the SEVER function to fully dismantle the control blocks.

- If the communicating partner has not received the pending connection interrupt, the path terminates without intervention from the communicating partner.

Note: CP system code (aside from IUCV support) cannot use the SEVER ALL function.

TEST COMPLETION Function

The source communicator executes the TEST COMPLETION function to complete a communication. The procedure includes:

- Dequeuing the MSGBLOK of the completed message from the source communicator's REPLY queue

- Destroying the dequeued MSGBLOK.

Data is not moved into the answer buffer by the TEST COMPLETION function. However, if the REPLY function is used with the DATA=PRMMSG option, the eight bytes of data will appear in the TEST COMPLETION parameter list.

TEST COMPLETION function specifies which message is to be processed; the message is identified by:

- Message ID
- Path ID
- Source message class.

Note: Message id and message class are not recognized for APPC/VM messages.

If the message ID is not specified, then any path ID and source message class combination is valid for the TEST COMPLETION function. The flag field of the TEST COMPLETION parameter list indicates the type of message description to be used. If no parameter list search fields are specified, the first REPLY queue message is presented. If the specified message is found, then along with the setting of a normal condition code, the following parameter list fields are stored:

- Message ID
- Path ID
- Flags
- Audit trail
- Message tag
- Source message class.

Note: Message ID, message tag, and source message class are not stored for APPC/VM messages.

If the specified message is not found, just a condition code is set. Note that the TEST COMPLETION function clears the pending message complete external interrupt for the REPLY queue message.

Note: CP system code (aside from IUCV support) cannot use the TEST COMPLETION function.

TEST MESSAGE Function

The virtual machine communicator invokes the TEST MESSAGE function to determine whether or not messages or replies are pending. The virtual machine enters a wait state when neither message or replies are pending.

The virtual machine communicator uses the TEST MESSAGE function to poll for IUCV current messages and/or replies as well as to wait for future messages and/or replies. An APPC/VM message pending on a path which is not in RECEIVE state is ignored by the TEST MESSAGE function.

Use of the TEST MESSAGE function allows an instruction stream to poll for a message and, simultaneously, to enter a wait state. If, during a wait state, an IUCV message and/or reply pending occurs, the virtual machine communicator reinvokes the TEST MESSAGE function to continue processing and a proper condition code is sent to the initiator.

Because IUCV messages are also presented as external interrupts, the TEST MESSAGE function introduces the anomaly of identifying IUCV messages simultaneously through the external interrupts and through the TEST MESSAGE function's polling capability. If the PSW external interrupt mask bit or the SET MASK function are disabled for IUCV messages and replies, the TEST MESSAGE function may be used to poll, and the DESCRIBE, RECEIVE, and TEST COMPLETION functions used to receive all information about IUCV messages and replies. This polling reduces external interrupt handling overhead.

The condition code setting indicates that there is at least one message and/or reply in the SEND queue or the REPLY queue. Status information is forthcoming when you execute the DESCRIBE function and the TEST COMPLETION function. Note that the condition code setting indicates the TEST MESSAGE function completion status.

Note: CP system code (aside from IUCV support) cannot use the TEST MESSAGE function.

Return codes applicable to IUCV are described in the *VM System Facilities for Programming.*

IUCV 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.
- The sum total of all CP system service connections cannot be greater than 4,096.

IUCV 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') with a subtype field to indicate exactly which IUCV function was invoked. *All* uses of IUCV, whether invoked from a virtual machine or from CP system code, 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 flag byte indicates whether this address is to be interpreted as a real address (when invoked from CP) or a virtual machine 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 the TRACFLG3 field of the PSA.

For IUCV functions that invoke other functions, the secondary functions are also recorded as having been invoked from CP. Examples of these secondary functions are:

- Retrieve Buffer generates a Sever ALL.
- Sever generates a Reject for each incoming outstanding message and a purge for each outgoing outstanding message.
- A Connect to a CP system service causes control to go to that service will usually invoke the Accept function.
- The IUCV support invokes the Test Completion function to dequeue messages intended for the CP system.

IUCV External Interrupts

Prior to establishing any connections, the virtual machine must invoke the Declare Buffer function to indicate to IUCV where data associated with an external interrupt is to be stored.

There is one external interrupt type for external interrupts generated by IUCV. This external interrupt type is X'4000'. When an IUCV external interrupt is reflected to the virtual machine, the interrupt code is stored for the virtual machine and an 'External Interrupt Buffer' is stored at the address specified in the DECLARE BUFFER function. One field of this buffer, IPTYPE, is an external interrupts subtype to indicate exactly why the external interrupt occurred. The possible codes and their meanings are as follows:

- 01 - IUCV Connection Pending
- 81 - APPC/VM Connection Pending
- 02 - IUCV Connection Complete
- 82 - APPC/VM Connection Complete
- 03 - IUCV Path Severed
- 83 - APPC/VM Path Severed
- 04 - Path Quiesced
- 05 - Path Resumed
- 06 - Incoming Priority Reply
- 07 - Incoming Nonpriority Reply
- 87 - Function Complete
- 08 - Incoming Priority Message
- 88 - SENDREQ Interrupt
- 09 - IUCV Incoming Nonpriority Message
- 89 - APPC/VM Incoming Message.

While the Connect, Accept, Sever, Quiesce, and Resume functions always cause a pending external interrupt to be queued for the target virtual machine or passed to a CP service, incoming messages and incoming replies can be fielded by the target virtual machine as either external interrupts or by the satisfaction of the Describe or Test Completion functions.

When a virtual machine executes a Send, a pending external interrupt of subtype 08, 09, 88, or 89 is queued for the target virtual machine. If the target virtual machine is both enabled for external interrupts (Bit 7 in the virtual PSW is set to 1) and enabled for messages (via the Set Mask function), then the external interrupt will be reflected. In the case of APPC/VM incoming message interrupts, the corresponding path must be in RECEIVE state before the interrupt is presented. If either condition is not met, the external interrupt will remain queued. If the target virtual machine is not enabled but instead executes the Describe function, the information about the pending message will be returned in the parameter list and the pending external interrupt for that particular message will be cleared.

The condition of being enabled for IUCV messages and issuing a Describe will cause unpredictable results. In a similar manner, the condition of being enabled for IUCV replies and issuing a Test Completion will cause unpredictable results. Although it is unpredictable as to whether the external interrupt is presented or the IUCV function (Describe or Test Completion) is satisfied, it is never the case that both the external interrupt and the IUCV function completion will occur for the same message/reply.

All IUCV external interrupts are controlled by the external mask bit in the virtual PSW (bit 7) and the submask bit in control register zero (bit 30).

There are separate additional mask bits for IUCV external interrupts that can be enabled and disabled by the Set Mask function. Five mask bits are defined for use by the Set Mask function. These mask bits are used to separately mask incoming priority messages, incoming nonpriority messages, incoming priority replies, incoming nonpriority replies, incoming APPC/VM messages, incoming APPC/VM SENDREQ interrupts, APPC/VM function complete interrupts, and IUCV control interrupts of subtypes 01, 81, 02, 82, 03, 83, 04, and 05. When both the external interrupt mask and the appropriate Set Mask bits are enabled, the external interrupt can occur.

The SETCMASK function lets you set masks for the individual IUCV control interrupts. The IPCMASK field specifies which of the 5 types of IUCV control interrupts for the virtual machine will be enabled. These interrupts include: Connection Pending, Connection Complete, Path Severed, Path Quiesced, and Path Resumed.

The SETMASK function is interrogated before the SETCMASK function mask. If you specify that all control interrupts are disabled, by using the SETMASK function, then the SETCMASK settings are not interrogated. If you specify that all control interrupts are enabled, by using the SETMASK function, then the SETCMASK settings are interrogated to determine how to handle the individual types of control interrupts.

After IUCV initialization and until you issue the SETMASK and SETCMASK function, all IUCV submask bits are on, enabling all IUCV external interrupts.

When the virtual machine has completed all communications, the virtual machine may invoke the Retrieve Buffer function to cause IUCV to stop using the external interrupt buffer and prevent further IUCV communication.

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.

The order of reflection for IUCV external interrupts is as follows:

- Control interrupts (Subtype X'01', X'81', X'02', X'82', X'03', X'83', X'04', X'05') in first-in-first-out (FIFO) order
- Priority Replies (Subtype X'06')

- Nonpriority Replies (Subtype X'07')
- Priority Messages (Subtype X'08') and SENDREQ (Subtype X'88')
- Nonpriority Messages (Subtype X'09') and APPC/VM Messages (Subtype X'89').

IUCV Control Blocks and Data Areas

Figure 23 on page 85 shows the relationships between the various IUCV control blocks and data areas. IUCV identifies and describes a communicator with an IUCVBLOK. The Communication Control Table (CCT), part of the IUCVBLOK, contains a Path Description Entry (PDENT) for each path defined for the communicator. There is a PDENT in the source CCT and a different PDENT in the target CCT for each path defined. Each of these PDENTs is identified, or named, by a Path Description Identifier (PDID or Path ID). There is a PDID for the source communicator's view of a path (its PDENT for the path) and another PDID for the target communicator's view of the same path (its PDENT). At the interface to IUCV, there is no relationship assumed between the two PDID values. A particular communicator can address a path only by that communicator's PDID.

Messages are represented by Message Blocks (MSGBLOKs). A MSGBLOK is created when a communication is initiated and is destroyed when a communication is terminated. A message, and its representation as a MSGBLOK, is fully identified by three values. These values are the PDID, the Message Class, and the Message ID. The source and target communicators each have their own description of a particular message.

A MSGBLOK represents an active communication and is chained onto one of the queues anchored in a CCT. A CCT contains 3 queues. These queues keep track of communication status. Queued MSGBLOKs can be handled FIFO, enqueued by priority, and dequeued by value.

The Send Queue (SENDQ) is defined for target communicators. It is the queue of MSGBLOKs that have been created by the source communicator but not yet accepted by the target communicator. The Receive Queue (RECQ) is also defined for target communicators. It is the queue of MSGBLOKs that have been accepted by the target communicator, but to which the target communicator has not yet replied. The Reply Queue (REPLYQ) is defined for source communicators. It contains those MSGBLOKs that have been replied to by a target communicator, but which have not yet been terminated.

A normal communication uses a MSGBLOK in a predefined manner. The MSGBLOK moves in sequence from a SENDQ, to an RECQ, and finally to a REPLYQ.

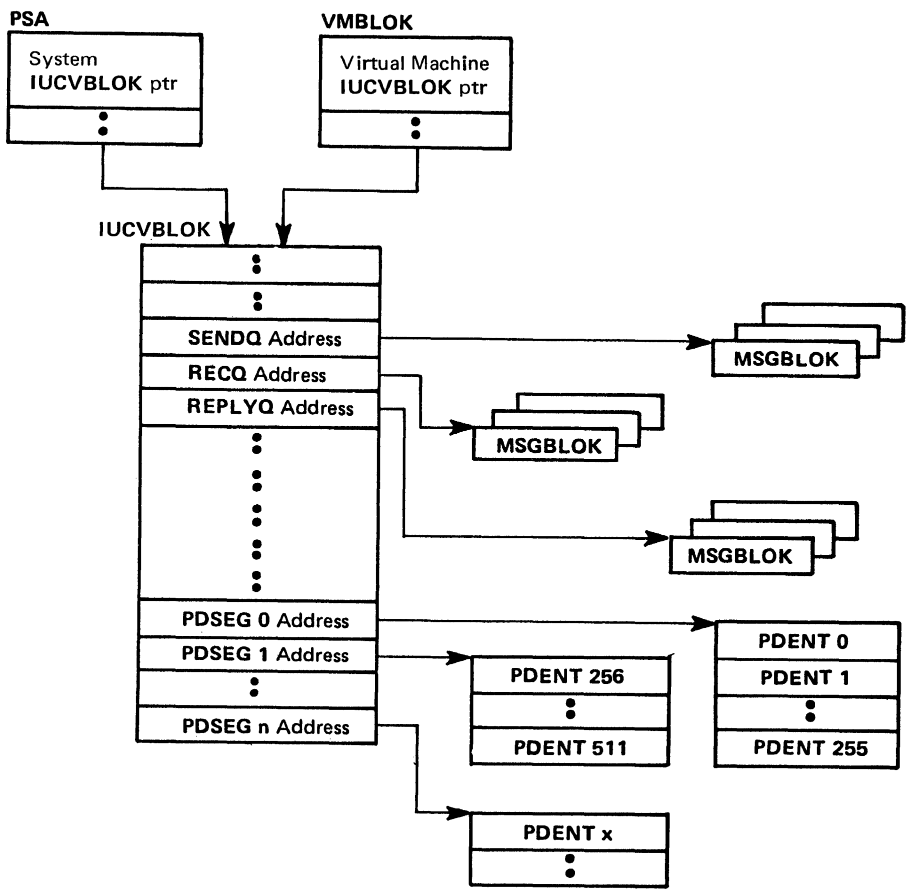


Figure 23. IUCV Control Block Relationships

VM/VS Handshaking

The VM/VS Handshaking feature provides a communication path between CP and virtual machine operating systems that makes each system control program aware of certain capabilities or requirements of the other.

The following is a discussion of VM/VS Handshaking as it relates to OS/VS1. Functions of VM/VS Handshaking incorporated in the VM/SP control program are available and applicable to any operating system that can be system generated to use this VM/SP function.

VM/VS Handshaking for OS/VS1 performs the following functions:

- Closes CP spool files when the VS1 job output from its DSO, terminator, and output writer is complete
- Processes VS1 pseudo page faults
- Provides an optional nonpaging mode for VS1 when it is run in the VM/SP environment.

When a VS1 virtual machine with the handshaking feature is loaded (via IPL), its initialization routines determine whether the handshaking feature should be enabled. First, VS1 determines if it is running under the control of VM/SP by issuing a STIDP (Store Processor ID) instruction. STIDP returns a version code; a version code of X'FF' indicates VS1 is running with VM/SP. If VS1 finds a version code of X'FF', it then issues a DIAGNOSE (X'00') instruction to store the VM/SP extended-identification code. If an extended-identification code is returned to VS1, VS1 knows that VM/SP supports handshaking; if nothing is returned to VS1, VM/SP does not support handshaking. At this time or any time after IPL, the operator of the VS1 virtual machine can issue the CP SET PAGEX ON command to enable the pseudo page fault handling portion of handshaking. If the VS1 virtual machine is in the nonpaging mode and, if the pseudo page fault handling is active, full handshaking support is available.

Because the VS1 system does no paging, any ISAM programs run under VS1 are treated by VM/SP as though they are running in an ADDRSPC=REAL partition. Therefore, the ISAM option is required for the VS1 machine to successfully execute the ISAM program.

Closing CP Spool Files

If the handshaking feature is active, VS1 closes the CP spool files when its job output from the DSO, terminator, and output writer is complete. Once the spool files are closed, VM/SP processes them and they are sent to the real printer or punch. During its job termination processing, VS1 issues a DIAGNOSE (X'08') instruction to pass the CP CLOSE command to VM/SP for each CP spool file.

Pseudo Page Faults

A page fault is a program interruption that occurs when a page marked `not in storage` is referred to by an instruction with an active page. The virtual machine referring to the page is placed in a wait state while the page is brought into real storage. Without the handshaking feature, the entire VS1 virtual machine is placed in page wait by VM/SP until the needed page is available.

However, with the handshaking feature, a multiprogramming (or multitasking) VS1 virtual machine can dispatch one task while waiting for a page request to be answered for another task. VM/SP passes a pseudo page fault (program interrupt X'14') to VS1. When VS1 recognizes the pseudo page fault, it places only the task waiting for the page in page wait and can dispatch another task.

When a page fault occurs for a VS1 virtual machine, VM/SP checks that the pseudo page fault portion of handshaking is active and that the VS1 virtual machine is in EC mode and enabled for I/O interruptions. Then, VM/SP reflects the page fault to VS1 by:

- Storing the virtual machine address that caused the page fault at location X'90' (the translation exception address).
- Indicating a program interruption (interrupt code X'14') to VS1
- Removing the VS1 virtual machine from page wait and execution wait

When VS1 recognizes program interruption code X'14', it places the associated task in wait state. VS1 can then dispatch other tasks.

When the requested page becomes available in real storage, VM/SP indicates the same program interruption to VS1, except that the leftmost bit in the translation exception address field is set on to indicate completion. VS1 removes the task from page wait; the task is then eligible to be dispatched.

VS1 Nonpaging Mode

When VS1 runs under the control of VM/SP, it executes in nonpaging mode if:

- Its virtual storage size is equal to the size of the VM/SP virtual machine.
- Its virtual machine size is at least 1024K bytes and no more than 4096K bytes. For VS1 Release 6, the maximum size is 16,384K bytes.
- The VM/VS Handshaking feature is available.

When VS1 executes in nonpaging mode, it uses fewer privileged instructions and avoids duplicate paging. The VS1 Nucleus Initialization Program (NIP) fixes all VS1 pages to avoid the duplicate paging.

Note: The working set size may be larger for a VS1 virtual machine in nonpaging mode than for one in paging mode.

Miscellaneous Enhancements

A VS1 virtual machine with the handshaking feature avoids many of the instructions or procedures that would duplicate the function that VM/SP provides. For example, VS1 avoids:

- Insert storage key (ISK) instructions and uses a key table
- Seek separation for 2314 direct access devices
- ENABLE/DISABLE sequences in the VS1 I/O Supervisor (IOS)
- Test channel (TCH) instructions preceding Start I/O (SIO) instructions.

Chapter 6. CP Interruption Handling

Interruption processing occurs within the CP environment. More than 30 modules control the process of interrupting events brought about by CP or virtual machine activity. Each module handles a particular I/O device or class or a function of CP, (for example: timers, paging, SVCs). For an overview of interruption handling, see Figure 24 on page 90.

TYPE	MODULE
SVC	DMKSVCIN
External	DMKEXTIN
Machine Check	DMKMCHIN
I/O	DMKIOTIN
Program Check	DMKPRGIN

Interrupt Handler Modules

Interrupt From	Action/Module
Unknown channel	Ignored – DMKDSPCH
Unsolicited device end	Build IOBLOK
and for:	
Console	DMKCNSIN
3270s on bisync lines	DMKRGA or DMKRGB
Local 3270, 3158, and 3066 consoles	DMKGRF
Unit record, real spooling	DMKRSPEX
Solicited device end	DMKSTKIO
Channel error	DMKCCHNT
Monitor tape I/O operation	DMKMONIO
Dedicated device error – DASD	DMKASER, DMKDADER or DMKDAUER
Dedicated device error – Tape	DMKTAPER
3270 bisync line and channel errors	DMKBSC
Recoverable errors	DMKSTKIO
Unrecoverable errors	DMKIOERR

I/O Interrupt Handler (DMKIOT) Actions

Reason for Program Check	Module
Normal paging	DMKPTRAN
Paging – virtual machine in EC mode	DMKVAT
Supervisor State	DMKDMP
Privileged instruction	DMKPRVLG or DMKFPS
DIAGNOSE	DMKHVC
Timers	DMKTMR
Virtual Machine I/O	DMKVSIEX
console	DMKVCNEX
unit record, virtual spooling	DMKVSPEX

Program Check Interrupt Handler (DMKPRG) Actions

Figure 24. Overview of Interruption Handling

Program Interruption

Program interruptions occur in two states. If the CPU is in the supervisor state, the interruption indicates a system failure in the CP nucleus and causes a system abnormal termination. If the CPU is in the problem state, a virtual machine is in execution. If the program interruption indicates that the Dynamic Address Translation (DAT) feature has an exception, a virtual machine issued a privileged instruction, or a protection exception occurred for a shared segment system, CP takes control and performs any required processing to satisfy the exception. Usually, the interruption is not apparent to the virtual machine. Most other program interruptions result from virtual machine processing and are reflected to the virtual machine for handling.

When a program interruption occurs, the program interruption handler (DMKPRG) is entered. Program interruptions can result from:

- Normal paging requests
- A paging request by a virtual machine in EC mode (virtual relocate mode)
- Privileged instructions
- PER event
- Program errors
- Monitor calls.

For information about paging requests, see "Allocation Management" on page 135.

Privileged Instructions

If a program interruption is caused by the virtual machine issuing a privileged instruction when it is running in supervisor state, DMKPRVLG or DMKFPS obtains the address of the privileged instruction and determines the type of operation requested. If the virtual machine was running in problem state, the interruption is reflected back to the virtual machine.

I/O Privileged Instructions

DMKPRVLG transfers control to the virtual I/O executive program (DMKVSIEX).

I/O Interruption

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

Missing Interrupt Handler

A missing interrupt condition exists when an I/O device fails to return an interrupt to the control program after a specified time interval. The missing interrupt handler automatically monitors system I/O activity to detect this condition and to attempt to correct it.

To detect missing interrupts, at system initialization module DMKCPI schedules TRQBLOKs to enter module DMKDID. DMKDID tests the RDEVBUZY and RDEVSCHD flag bits in each REVBLOK. It sets RDEVMID to one when it scans the RDEVBLOKs and finds RDEVBUZY or

RDEVSCHD ON, indicating that a device interrupt is pending for this interval.

Then the first-level interrupt handler, DMKIOT, resets the flags when the device returns an interrupt. If both flags are on when DMKDID again receives control, a missing interrupt condition exists.

If you want CP to take corrective action and if you have set MIH on, CP attempts to simulate the interrupt. If this is unsuccessful, send an IFCC to the guest virtual machine. A record is written to the error recording area and an informational message is sent to inform the system operator of the missing interrupt and to indicate whether or not it was cleared.

If MIH was set off, CP pursues no corrective action. However, message DMKDID546I will be written alarming the operator that a missing interrupt was detected, and a record will be written to the error recording area.

MIH can be set on by specifying an option in the directory or by issuing the class G SET command.

If you enable tracing activity, CP traces the event and records it as trace table entry `X'19'`. See *VM Diagnosis Guide* for more information about tracing.

Interrupt timing varies widely among devices. Certain devices are more critical than others. To allow greater flexibility for monitoring I/O activity, CP specifies a different time interval for each class of device. You can change the IBM supplied defaults (provided in module DMKSYS) in two ways. You can change the values supplied in the SYSMIH macro statement in DMKSYS and reassemble DMKSYS, or you can use the SET MITIME command.

The default intervals and the devices monitored are:

CLASDASD and CLASFBA	15 seconds
CLASGRAF	30 seconds (except TYP1053 and TYP328X)
CLASTAPE	10 minutes
CLASURI and CLASURO	1 minute (except TYP3800 and TYP3289E)
Miscellaneous devices	12 minutes

Miscellaneous devices include MSS devices, graphic devices: TYP1053 and TYP328X, and UR output devices: TYP3800 and TYP3289E.

The SET MITIME command dynamically changes the intervals specified in DMKSYS. Time intervals used in the SET MITIME command remain in effect until you issue another MITIME command, or until the system is reloaded (through IPL). SET MITIME is described in the *VM/SP CP Command Reference*. See the *VM/SP Planning Guide and Reference* for a description of the SYSMIH macro statement. For the method of operation, refer to "Monitoring I/O Activity" on page 118.

Non-I/O Privileged Instructions

DMKPRVLG simulates valid non-I/O privileged instructions and returns control to DMKDSPCH. For invalid non-I/O privileged instructions, the routine sets an invalid interruption code and reflects the interruption to the virtual machine. For the privileged instructions (SCK, SCKC, STCKC, SPT, and STPT) that affect the TOD clock, CPU timer, and TOD clock comparator, control is transferred to DMKTMR by DMKPRVLG. Other instructions that are simulated are LPSW, SSM, SSK, ISK, IPTE, TB, TPROT, and DIAGNOSE.

DMKFPS attempts to simulate certain non-I/O privileged instructions and to redispatch the virtual machine. If DMKFPS cannot fully simulate the privileged instruction, DMKPRVLG simulates it. Instructions that DMKFPS simulates are: PTLB, LCTL, TCH, STNSM, STOSM, SPT, STPT, IPTE, TPROT, and SIOP. Other instructions that are simulated are LPSW, SSM, SSK, ISK, IPTE, TB, TPROT, and DIAGNOSE.

System/370 EC mode non-I/O privileged instruction simulation includes the following:

Code	Definition
SCK	Set Clock
SCKC	Set Clock Comparator
STCKC	Store Clock Comparator
SPT	Set CPU Timer
STPT	Store CPU Timer
STNSM	Store and AND System Mask
STOSM	Store and OR System Mask
STIDP	Store CPU Identification
STIDC	Store Channel Identification
LCTL	Load Control
STCTL	Store Control
LRA	Load Real Address
RRB	Reset Reference Bit
PTLB	Purge Table Look-aside Buffer
IPK	Insert PSW Key
SPKA	Set PSW Key From Address
IPTE	Invalidate Page Table Entry
TPROT	Test Protection
TB	Test Block

Machine Check Interruption

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

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

- System termination.
- In attached processor or multiprocessor VM/SP configurations, a processor or an attached processor is varied offline (system converts to uniprocessor mode).
- One or more channels are marked offline.
- Virtual user running at the time of error is terminated.
- 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 set up well enough 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 Interruption

When an SVC interruption occurs, the SVC interruption routine (DMKSVCIN) is entered. If the machine is in the problem state, DMKSVC branches to DMKSVDIN. DMKSVDIN takes the following action:

- If the interruption was the result of an ADSTOP (SVC code X'B3'), the message ADSTOP AT XXXXX is sent to the user's terminal, the overlaid instruction is replaced, and the virtual machine is placed in console function mode (CP mode) via DMKCFMBK.
- If the interruption was the result of an error recording interface (SVC 76), DMKSVD checks for valid parameters and passes control to DMKVER to convert virtual device addresses in the error record to real device addresses. The actual recording is accomplished in DMKIOE and DMKIOF. If recording is not possible, the interrupt is reflected back to the virtual machine.
- If the virtual machine's page 0 was not in real storage,then all general and floating-point registers are saved, the user's VMBLOK is flagged as being in an instruction wait, and control is transferred (via GOTO) to DMKPRGRF to reflect the interruption to the virtual machine.

- If the virtual machine's page 0 is in main storage, an appropriate SVC old PSW is stored in the user's page 0 and the interruption is reflected to the virtual machine, bypassing unnecessary register saving (fast reflection). If the new virtual PSW indicates a mode or enablement change, all registers are saved in the VMBLOK and control is transferred to DMKDSPB for PSW validation.

If the machine is in the supervisor state, then DMKSVC determines the SVC interrupt code and a branch is taken to the appropriate SVC interruption handler.

SVC 0	Impossible condition or terminal error. The SVCDIE routine initiates an abnormal termination by using the DMKDMPDK routine.
SVC 4	Reserved for IBM use.
SVC 8	A link request that transfers control from the calling routine to the routine specified by register 15. The SVCLINK routine sets up a new save area, and then saves the caller's base register in register 12 and save area address in register 13, and the return address (from the SVCOPSW) in the new save area. If the called routine is within the resident CP nucleus, SVCLINK places its address in register 12 and branches directly to the called routine. If the called routine is in a pageable module, a TRANS macro is performed for register 12 to ensure that the page containing the called routine is in storage. Upon return from the TRANS execution, the real address of the pageable routine is placed in register 12 and SVCLINK branches to the called routine. The real storage location of DMKCPE is the end of the resident CP nucleus. Any modules loaded at a higher real storage address are defined as pageable modules. If bit zero of register 15 is on when DMKSVC is entered, then the caller has requested AFFINITY. DMKSVC turns on a bit in the save area passed to the caller to indicate that control is to be returned to the caller on the same processor on which it was running before issuing the SVC. It is not ensured that control will be retained by the initiating processor throughout the called operation, but only that final return will occur on the initiating processor.
SVC 12	A return request that transfers control from the called routine to the calling routine. The SVCRET routine is invoked. If the routine that issued the SVC 12 is pageable, then DMKPTRUL is called to unlock the page. SVCRET then restores registers 12 and 13 (addressability and save area address saved by SVCLINK), places the user's return address (also saved in this area) back into the SVCOPSW, and returns control to the calling routine by loading the SVCOPSW.
SVC 16	Releases current save area from the active chain (removes linkage pointers to the calling routine). The SVCRLSE routine releases the current save area by placing the address of the next higher save area in register 13 and returns control to the current routine by loading the SVCOPSW. This SVC is used by second

level interrupt handlers to bypass returning the first-level handler under specific circumstances. The base address field (register 12) in the save area being released is examined to determine if the bypassed routine is in a pageable module. If so, DMKPTRUL is called to unlock the page.

SVC 20 Obtain a new save area. The SVCGET routine places the address of the next available save area in register 13 and the address of the previous save area in the save area pointer field of the current save area.

SVC 24 In attached processor mode, SVC 24 causes the instructions following the SVC to be executed by the main processor. This SVC is used only via the SWITCH macro to force processing to continue on the main processor (the processor capable of performing I/O). If the SWITCH macro determines that the code is currently running on the main processor then the SVC is not issued.

In multiprocessor mode, the SWITCH macro is coded with the 'PROC' operand. SVC 24 causes the instructions following the SVC to be executed on the processor specified by the PROC = operand.

Save areas are initially set up by DMKCPI for use by the SVC linkage handlers. There is one list of save areas per processor. The number of save areas to be pre-allocated is determined by DMKCPI based upon the model number of the processor and upon whether or not the system was AP or MP generated.

External Interruption

Timer Interruption

If DMKPSAEX is entered because of a timer interruption, the state of the machine must be determined. If the virtual machine was in wait state, control is transferred to DMKDSPCH, and the virtual machine stays idle until another interruption occurs. If the virtual machine is in problem state, the address of the current user's VMBLOK is obtained from RUNUSER. The user's current PSW (VMPSW) is updated from the external interruption old PSW, the address of the current VMBLOK is placed in register 11, and control is transferred to DMKDSPCH. For additional information about timers, see "Virtual Timer Maintenance" on page 106.

External Interruption

If DMKPSAEX is entered because the operator pressed the console interrupt button (INTERRUPT), a CPEXBLOK is stacked to do the following:

- If Multiprocessor mode (MP mode), continue processing on the processor which has an I/O path to the operator's console.
- Reference the current system operator's VMBLOK (DMKSYSOP).
- Disconnect this virtual machine.

The operator can now log on from another terminal. Pressing the console interrupt button activates an alternate operator's console.

Note: If this interrupt comes from the attached processor, it is ignored.

See "Multiprocessor External Interrupts" on page 234 for a discussion of external interrupts that occur in attached processor or multiprocessor mode.

Extended Virtual External Interruptions

To reflect external interruptions to a virtual machine, DMKDSPE queues an XINTBLOK on a chain pointed to by VMPXINT in the VMBLOK. The XINTBLOKs are chained sequentially by the XINTSORT field that contains the collating number of the pending interruption. If more than one interruption has the same collating number, the interruption codes are ORed together in the XINTCODE field for possible simultaneous reflection.

When a virtual machine is enabled for external interruptions, the XINTBLOK queue for that machine is searched for an eligible block. An XINTBLOK is eligible for reflection if one or more bits of the XINTMASK field match the bits in the rightmost halfword of control register 0. If the interruption was an interruption such as CPU timer or clock comparator, the block is left chained because reflection does not reset these interruptions. If the reflected interruption(s) does not represent all those coded in the XINTMASK field, the block is left chained and only the interruptions that were reflected are reset. In all other conditions, the XINTBLOK is unchained and returned to free storage.

A special external interrupt, code X'4001' notifies a virtual machine of a pending Virtual Machine Communication Facility request. The XINTBLOK for this interrupt is set up with an XINTSORT field of X'7FFFFFFF', the lowest priority.

System Support

Storage Protection

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 unauthorized 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 the processor accesses storage, and protection applies, the protection key of the current PSW is used as the comparand. The protection key of the processor is bit positions 8 through 11 of the PSW.

If the processor access is prohibited because of a protection violation, the operation is suppressed or terminated, and a program interruption for a protection exception takes place.

When the reference is made to 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 in bit positions 0-3 of the CAW and is recorded in bit positions 0-3 of the CSW stored as a result of an I/O operation. If channel access is prohibited, the CSW stored as a result of the operation indicates a protection-check condition.

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

To use fetch protection, a virtual machine must execute the set storage key (SSK) instruction. The fetch protection bit in the storage key refers to the data area that is to be protected. CP subsequently:

- Checks for a fetch protection violation when handling privileged and nonprivileged instructions.
- Saves and restores the fetch protection bit (in the virtual storage key) when writing and recovering virtual machine pages from the paging device.
- Checks for a fetch protection violation on a write CCW (except for spooling or console devices).

A special case of storage protection occurs when the CMS nucleus resides in a protected shared segment. The CMS nucleus may be protected and still be shared by many CMS users. After a virtual machine has used a protected shared segment, the pages are checked for changes. If any pages have been changed, the user gets placed in console function mode after receiving an error message, and the changed page is returned to CP free storage.

Storage Validation

At VM/SP load, the loader (DMKLD00E) uses the TB instruction as it relocates itself to the high-end of storage and as it loads the VM/SP system modules into storage. The VM/SP nucleus must reside in contiguous storage. If an unusable or non-addressable frame is detected within the area reserved for the nucleus, the system load is terminated with a disabled wait state code X'AAAAAA'. There is one exception. Non-addressable frames and frames having errors encountered in the virtual=real area do not cause a disabled wait state at VM/SP load. Instead, informational messages are sent to the system operator. This presents a special consideration for virtual machine operating systems running V=R . The V=R guest should use appropriate storage techniques to validate V=R storage and avoid machine errors in real storage.

VM/SP initialization routines DMKCKP, DMKSAV, and DMKSTA issue the TB instruction to determine the status of every frame of real storage. If a non-addressable frame or a frame containing errors is detected within the area reserved for the nucleus (excluding the V=R area), system initialization is terminated with a disabled wait state code X'14'. Storage frames reserved for the V=R area are not validated at VM/SP initialization. V=R area frames are validated only at VM/SP load time as described above. Non-addressable and invalid frames encountered outside the area occupied by the VM/SP modules are identified to the system operator by a series of informational messages.

Executing the Pageable Control Program

Calls to pageable routines are recognized at execution time by the SVC 8 linkage manager in DMKSVC. For every SVC 8, the called address (in the caller's GPR15) is tested to see if it is within the resident nucleus. If it is less than DMKCPEND and greater than DMKSLC, the called routine's base address is placed in GPR12 and control is passed to the called routine in the normal way. However, if the called address is above DMKCPEND or below DMKSLC, the linkage manager issues a TRANS macro, requesting the paging manager to locate and, if necessary, page-in the called routine. The TRANS is issued with LOCK option. Thus, the lock count associated with the called routine's real page indicates the responsibility count of the module.

- When the module is called, the count is incremented.
- When the routine exits via SVC 12, the count is decremented.

When the count reaches zero, the pageable routine is unlocked and is eligible to be paged out of the system. However, because all CP pageable modules are reenterable, the page is never swapped out, but when the page is stolen, it is placed directly on the free page list.

Because unlocked pageable routines participate in the paging process in a manner similar to user virtual storage pages, the least recently used approximation page selection algorithm tends to make highly used control program routines resident, even when they are not locked. The called routine is locked into real storage until it exits. Thus, it can request asynchronously scheduled functions, such as I/O or timer interrupts, as long as it dynamically establishes the interruption return address for the requested operation and does not give up control via an EXIT macro prior to receiving the requested interruption.

Addressability for the module, while it is executing, is guaranteed because the CALL linkage loads the real address of the paged module into GPR12 (the module base register) prior to passing control. If all addressing is done in a base/displacement form, it is not apparent that the module is executing at an address different from that at which it was loaded. Although part of CP is pageable, it never runs in relocate mode. Thus, the processor is not degraded by the DAT feature being active, and no problems occur because of handling disabled page faults.

System Support Modules

The system support modules provide CP with several common functions for data conversion and control block scanning and verification. Most of the routines are linked to via the BALR option of the CALL macro, and make use of the BALRSAVE and TEMPSAVE workareas in DMKPSA. Two exceptions are the virtual and real I/O control block scan routines DMKSCNVU and DMKSCNRU. These routines do not alter the contents of the BALRSAVE area, and hence may be called by another low-level BALR routine.

Control Register Usage

Every IBM System/370 processor provides the program with 16 logical control registers (logical registers since the number that are active depends on the features installed in the machine at any one time) that are addressable for loading and storing from basic control (BC) mode. VM/SP provides only a single control register, control register zero, for normal virtual machines, and for processing systems that do not require the full set of registers (for example, CMS, DOS, or other operating systems for System/360).

Any user whose virtual machine operating system requires the use of control registers other than control register zero, can request the full set of 16 registers by specifying the ECMODE option in the VM/SP directory entry for his virtual machine.

A virtual machine, which utilizes any System/370 features that use the control registers, requires the ECMODE option. Some of these features are expanded timer support of the System/370 CPU timer, clock comparator, etc, the virtual relocate mode and its instructions, RRB, LRA, PTLB, virtual monitor calls, virtual Program Event Recording (PER), etc.

Restrictions and Conventions for Pageable CP Modules

Pageable CP modules must observe the following restrictions and conventions when they are designed and coded:

- The module must be entered by the standard SVC 8 CALL linkage. Modules entered by BALR or GOTO cannot be pageable. The module must return to its caller by SVC also.
- The module cannot contain any A- or V-type address constants that point to locations within itself or within other pageable modules, and it cannot contain any CCWs that contain data addresses within themselves. The only exceptions are address constant literals generated as the result of calls to other modules (because these addresses are dynamically relocated at execution time, they must be resolved by the loader to the loaded address of the called module) and a pageable module that locks itself into storage. In practice, this restriction means that data or instructions within the pageable routine must be referenced via base/displacement addressing, and the address in register 15 for a CALL may not be generated by a LOAD ADDRESS instruction.
- The pageable module must be no more than 4096 bytes in length.

If the three above design and coding restrictions are adhered to, the CP module can be added to the existing pageable nucleus modules by utilizing the service routine, VMFLOAD, which is described in "VM/SP Maintenance Procedures" of the *VM/SP Service Routines Program Logic*. Additional information can be found in the *VM/SP Planning and System Generation Guide.*

Executable Pageable Modules

DMKACO
DMKALG
DMKALO
DMKAPI
| DMKAPS

| DMKAPT
| DMKAPU
| DMKAPV
| DMKAPW
| DMKAPX

| DMKAPY
| DMKAPZ
DMKATS
DMKBIO
DMKBLD

DMKBSC
DMKCDB
DMKCDM
DMKCDS
DMKCFC

DMKCFD
DMKCFF
DMKCFG
DMKCFH
DMKCFJ

DMKCFO
DMKCFP
DMKCFQ
DMKCFR
DMKCFS

DMKCFT
DMKCFU
DMKCFV
DMKCFW
DMKCFY

DMKCKS
DMKCKT
DMKCKV
DMKCLK
DMKCPB

DMKCPI
DMKCPJ
DMKCPM
| DMKCPN
DMKCPO

DMKCPP
DMKCPS
DMKCPT
DMKCPU
DMKCPV

DMKCPW
DMKCPX
| DMKCPY
| DMKCQC
DMKCQG

DMKCQH
DMKCQP
DMKCQQ
DMKCQR
DMKCQS

DMKCQT
| DMKCQU
DMKCQY
| DMKCRM
DMKCSB

DMKCSC
DMKCSF
DMKCSO
DMKCSP
DMKCSQ

| DMKCSR
DMKCST
DMKCSU
DMKCSV
| DMKCSW

| DMKCSX
DMKCVU
DMKDEF
DMKDEG
DMKDEI

DMKDIA
DMKDIB
| DMKDIF
DMKDRD
DMKEIG

DMKEPS
DMKERM
DMKGIO
| DMKGRA
DMKHPS

DMKHPT
| DMKHPU
DMKHVD
DMKHVE
| DMKHVF

| DMKIDR
DMKIDU
DMKIMG
DMKIOC
DMKIOF

DMKIOG
DMKIOH
DMKIOJ
DMKISM
DMKIUC

DMKIUG
DMKIUJ
DMKIUL
| DMKIUP
DMKJRL

DMKLNK
| DMKLNM
DMKLOC
DMKLOG
DMKLOH

DMKLOJ
DMKLOM
DMKMCC
DMKMCD
DMKMCI

DMKMHV
DMKMIA
DMKMID
DMKMNI
DMKMNJ

DMKMNT
DMKMON
DMKMOO
DMKMSG
DMKNEA

DMKNEM
DMKNES
DMKNET
DMKNLD
DMKNLE

| DMKNMT
DMKOPE
DMKPEI
DMKPEL
DMKPEN

DMKPEQ
DMKPER
DMKPET
DMKPGM
DMKQCP

| DMKREI
| DMKRPD
| DMKRPI
| DMKRPW
DMKRSE

DMKRSF
DMKRSQ
DMKRST
DMKSAV
DMKSBL

DMKSCO
DMKSEG
DMKSEP
DMKSEV
DMKSIX

DMKSNC
DMKSND
DMKSPK
DMKSPL
DMKSPM

DMKSPS
DMKSPT
DMKSRM
DMKSST
DMKSSV

DMKSTA
DMKSTP
DMKSTR
DMKTAP
DMKTAQ

DMKTCS
DMKTCT
DMKTDK
DMKTHI
DMKTOD

DMKTPE
DMKTRA
DMKTRC
DMKTRD
DMKTRM

DMKTRP
DMKTRT
DMKTRU
DMKTRX
DMKTTX

DMKTTY
DMKUDR
DMKUDU
DMKURS
DMKUSO

DMKUSP
DMKUSQ
DMKVBM
DMKVCA
DMKVCB

DMKVCH
DMKVDA
DMKVDB
DMKVDC
DMKVDD

DMKVDE
DMKVDF
DMKVDG
DMKVDR
DMKVDS

DMKVER
DMKVMC
DMKVMD
DMKVME
DMKVMG

DMKVMI
DMKVSU
DMKWRM
DMKWRN
DMKXAB

DMKXAD
DMKZTD

Executable Resident Modules

DMKACR
DMKACS
DMKCCD
DMKCCF
DMKCCH

DMKCCO
DMKCCS
DMKCCT
DMKCCW
DMKCFM

DMKCNS
DMKCVT
DMKDAD
DMKDAS
DMKDAU

DMKDGD
DMKDGF
DMKDID
DMKDMP
DMKDSB

DMKDSP
DMKENT
DMKEXT
DMKFPS
DMKFRE

DMKFRT
DMKGRC
DMKGRF
DMKGRH
DMKGRT

DMKHVC
DMKIOE
DMKIOQ
DMKIOS
DMKIOT

DMKIUA
DMKIUB
DMKIUE
DMKIUN
DMKIUS

DMKLOK
DMKMCH
DMKMCT
DMKMHC
DMKMSW

DMKOPR
DMKPAG
DMKPAH
DMKPGS
DMKPGT

DMKPGU
DMKPRG
DMKPRV
DMKPRW
DMKPSA

DMKPTR
DMKQCN
DMKQCO
DMKQCQ
DMKQVM

DMKRET
DMKRGA
DMKRGB
DMKRGC
DMKRGD

DMKRGE
DMKRNH
DMKRPA
DMKRSP
DMKSCH

DMKSCN
DMKSLC
DMKSSS
DMKSSU
DMKSTK

DMKSVC
DMKSVD
DMKTMR
DMKTRK
DMKUNT

DMKVAT
DMKVAU
DMKVCN
DMKVCW
DMKVCP

DMKVCQ
DMKVCR
DMKVCS
DMKVCT
DMKVCU

DMKVCV
DMKVCX
DMKVIO
DMKVMA
DMKVSC

DMKVSI
DMKVSJ
DMKVSP
DMKVSQ
DMKVSR

DMKVST
DMKVSV
DMKVSW
DMKVSX

Executable Modules During Initialization and Shutdown

DMKCKD
DMKCKF
DMKCKH
DMKCKM
DMKCKN

DMKCKP

Standalone Modules (Non-Pageable, Non-Resident)

DMKDDR
DMKDIR
DMKFMT
DMKOVR
DMKRND

DMKSAD
DMKSSP

Data Area Modules

In addition to the executable resident and pageable modules shown on the previous page, there are certain modules that only contain data areas and do not contain executable code. These modules are:

Resident Module	Contents
DMKCPE	Defines the end of the CP nucleus
DMKRIO	I/O device blocks
DMKSEQ	Printer separator logo
DMKSTD	Starting address of STDATA table
DMKSYS	System constants
DMKTBL	Terminal translate table
DMKTTZ	CCWs and data pointed to by certain CCWs for TTY terminals

Pageable Module	Contents
DMKBOX	Output separator table
DMKCMD	CP subcommand table
DMKFCB	3211-type Forms Control Buffer (FCB) load tables
DMKMES	CP message data module
DMKPIA	3289 Model 4 Font Offset Buffer (FOB) load tables
DMKPIB	3262 Universal Character Set Buffer (UCSB) load tables
DMKSNT	System name table
DMKSYM	System symbol table
DMKTBM	Terminal translate tables
DMKTBN	Terminal translate tables for APL/ASCII for TTY terminals
DMKUCB	3211 Universal Character Set Buffer (UCSB) load tables
DMKUCC	3203 Universal Character Set Buffer (UCSB) load tables
DMKUCS	1403 Universal Character Set (UCS) load tables

Virtual Timer Maintenance

The System/370 with EC mode provides the system user (both real and virtual) with four timing facilities. They are:

- The interval timer at main storage location X'50'
- The time-of-day clock
- The time-of-day clock comparator
- The CPU timer.

Real Timing Facilities

Before describing how CP maintains these timers for virtual machines, it is necessary to review how VM/SP uses the timing facilities of the real machine.

- The location X'50' interval timer is used only for time-slicing. The value placed in the timer is the maximum length of time that the dispatched virtual machine is allowed to execute. Because the BLIP function of CMS uses the interval timer (location X'50'), the use of STIMER can cause extra blips at the user's terminal. To avoid extra blips, issue the CMS command SET BLIP OFF.

- The time-of-day clock is used as a time stamp for messages and enables the scheduler to compute elapsed in-queue time for the dispatching priority calculation.

- The time-of-day clock comparator facility is used by CP to schedule timer-driven events for both control program functions and for virtual machines. A stack of comparator requests is maintained and as clock comparator interrupts occur, the timer request blocks are stacked for the dispatcher via calls to DMKSTKIO.

- The processor timer facility performs three functions:

  - Accumulates CP overhead
  - Detects in-queue time slice end
  - Simulates virtual processor timer.

  The accumulation of CP overhead is accomplished as follows. The VMTTIME field in the VMBLOK contains the total CP overhead incurred by the virtual machine; it is initialized to the maximum positive number in a doubleword, X'7FFFFFFF FFFFFFFF'. Whenever CP performs a service for a virtual machine, GR 11 is loaded with the address of the VMBLOK and the current value in VMTTIME is placed in the processor timer. When CP is finished with the service for that virtual machine, the processor timer, which has been decremented by the amount of processor time used, is stored back into VMTTIME. GR 11 is then loaded with a new VMBLOK pointer and the processor timer is set from the new VMTTIME field. The amount of CP overhead for a given virtual machine at any point in time is the difference between the maximum integer and the current value in the VMTTIME field.

  Since VMTTIME only accounts for supervisor state overhead, detection of in-queue time slice end is performed by the processor timer when the virtual machine is dispatched in the problem state. The VMTMOUTQ field in the VMBLOK is initialized to the amount of problem state time that the virtual machine is allowed to accumulate before being dropped from a queue. This initial value is set by the scheduler (DMKSCH) when the virtual machine is added to a queue and its value depends on the queue entered (interactive or noninteractive) and on the processor model. For example, the initial value of VMTMOUTQ for a user entering Q1 (interactive) on a Model 145 is 300 milliseconds, while for the same user entering Q2 (noninteractive) it is 2 seconds. Each time the user is dispatched, the value in VMTMOUTQ is entered into the processor timer; whenever the user is interrupted, the decremented processor timer is stored into VMTMOUTQ prior to being set from the new VMTTIME. When the problem state time slice has been exhausted, a processor timer interrupt occurs, the VMQSEND flag bit is set in the VMBLOK, and the scheduler drops the user from the queue. At each queue drop, the problem time used in-queue (the difference between

VMTMOUTQ and the initial value) is added to the total problem time field (VMVTIME) in the VMBLOK.

Virtual processor timer simulation is handled for EC mode virtual machines if the value in the virtual processor timer is less than that in VMTMOUTQ. In this case, the VMBLOK is flagged as `tracking processor timer` and a processor timer interrupt is interpreted as a virtual timer interrupt rather than as an in-queue time slice end.

Virtual Timing Facilities

Virtual location X'50' timers are updated by the elapsed processor time each time the dispatcher has been entered after a running user has been interrupted. The size of the update is the difference between the value of the timer at dispatch (saved in QUANTUM at location X'54') and the value of the timer at the time of the interruption (saved in QUANTUMR at location X'4C').

Virtual clock comparator requests are handled by the virtual timer maintenance routine, DMKTMR. They are inserted into the general comparator request stack and the virtual machine is posted when the interruption occurs.

Virtual clock comparator requests to set the virtual processor timer place the new value into the ECBLOK. Requests to store the new value update the ECBLOK field with the virtual processor time used since the last entry to dispatch and pass the value to the user. Requests to set the time-of-day clock are ignored.

A real interval timer or processor timer is one that runs when the virtual machine is executing or is in a self-imposed wait state (that is, the wait bit is on in the virtual PSW). A real timer does not run if the virtual machine is in a CP pseudo wait state (for example, page wait or I/O wait) or if the virtual machine can be run but is not being dispatched because of other user interaction. Real timers provide accurate interrupts to programs that depend on measurement of elapsed processor and/or wait time. They do not accurately measure wall time -- the TOD clock must be used for this function.

An EC mode virtual machine with the real timer option has both a real interval timer and a real processor timer. Real timer requests for waiting machines are maintained in the clock comparator stack. CPU timer requests are added to TOD clock value at the time that they are issued. Interval timer requests must have their units converted. In addition, if the virtual processor timer contains a large negative value, then a real timer request is scheduled to occur when the virtual processor timer becomes positive, so that the pending timer interruption can be unflagged. Comparator requests for real timer interruptions are inserted into the stack whenever a virtual machine enters a self-imposed wait. They are removed either when the virtual machine resumes execution or when it is forced (or places itself) into a pseudo wait.

I/O Management

I/O Supervisor

The module, DMKIOS, handles the I/O requirements of all system devices except the following terminals: 1052, 3210, 3215, 2150, 2741, and compatible teletypewriter devices. Scheduling and interruption handling for these devices is essentially a synchronous process and does not require the queuing and restart services of DMKIOS. This is handled by the module DMKCNS. For handling the I/O requirements of 3270 remote equipment, refer to "Remote 3270 Programming" on page 126.

Real I/O Control Blocks

To schedule I/O requests and control the activity of the I/O devices of the system, I/O control uses several types of control blocks. These blocks are separated into two basic types:

- Static blocks that describe the components of the I/O system
- The dynamic blocks that represent active and pending requests for I/O operations.

The I/O devices of the real system are described by one control block for each channel, control unit, and device available to the control program. For multiprocessor generated systems, two sets of real channel blocks are created. Units present but not represented by control blocks are not available for either user-initiated or CP-initiated operations.

Because all virtual machines are run in the problem state, any attempt to issue a SIO instruction results in a program interruption that indicates a privileged operation exception. This interruption is handled by CP's first level program interrupt handler, DMKPRGIN. It determines if the virtual machine was in virtual supervisor state (problem state bit in the virtual PSW is zero). If so, the instruction causing the interruption is saved in the VMBLOK for the virtual machine and control is transferred to the privileged instruction simulator, DMKPRVLG, via a GOTO.

DMKPRVLG determines if the privileged operation affects the virtual I/O configuration. DMKPRVLG simulates non-I/O privileged instructions (such as LPSW). If the instruction's operation code is from X'9C' to X'9F', control is transferred to DMKVSIEX.

After clearing the condition code in the user's VMBLOK, DMKSCNVU is then called to locate the virtual I/O blocks representing the I/O components (channel, control unit and device) addressed by the instruction. DMKVSIEX then branches to handle the request based on the operation requested.

In attached processor systems and multiprocessor systems, the I/O control blocks are protected by the I/O lock, a global spin lock.

Virtual I/O Requests

The virtual I/O interface maintained by CP provides to the software operating in the user's virtual machine, the condition codes, CSW status information, and interruptions necessary to make it appear to the user's virtual machine that it is in fact running on a real System/370. The virtual I/O interface consists of:

- A virtual I/O configuration for each active virtual machine that consists of a set of I/O control blocks that are maintained in the Control Program's free storage. This configuration is built at logon time from information contained in the user's directory file, and can be changed by the user or the system operator.
- A set of routines that maintain the status of the virtual I/O configuration.
- Other system routines that simulate or translate the channel programs provided by the user to initiate I/O on units in the real system's configuration.

Virtual SIO

With a SIO, the condition code returned from DMKSCNVU is tested to verify that all addressed components were located. If they were not, then a condition code of 3 (unit not available) is placed in the PSW and control returns to the dispatcher. Otherwise, the addresses of the appropriate virtual I/O control blocks are saved, and DMKVSIEX tests the status of the addressed I/O units by scanning the VCHBLOKs, VCUBLOKS, and VDEVBLOKs to locate the block that contains the status of the addressed subchannel. The subchannel status is indicated in:

- The VCHBLOK for a selector or block multiplexer channel
- The VCUBLOK for a shared selector subchannel on a byte multiplexer channel
- The VDEVBLOK for a nonshared subchannel on a byte multiplexer channel.

When the block containing the status is found, the status is tested. If the subchannel is busy or has an interruption pending, condition code 2 is placed in the virtual PSW. Otherwise, the subchannel is available and the device and the control unit are tested for interruption pending or busy. If either is found, condition code 1 is placed in the virtual PSW and the proper CSW status is stored in the virtual machine's page zero. If all components in the subchannel path are free, DMKVSIEX proceeds to simulate the SIO by locating and loading the contents of the virtual machine's CAW from virtual location X'48' and testing the device type of the unit addressed.

The device type is in the VDEVBLOK. If the device class code indicates a terminal or console, control is passed to the module DMKVCNEX with a GOTO. DMKVCNEX interprets and simulates the entire channel program, moving the necessary data to or from virtual storage and reflecting the proper interruptions and status bytes. When DMKVCNEX has finished, it passes control directly to the dispatcher, DMKDSPCH.

If the referenced device is a spooled unit record device, DMKVSIEX passes control to DMKVSPEX for additional processing. When control returns to DMKVSIEX, it passes control to DMKDSPCH.

If the device is not a terminal or a spooling device, the SIO is translated and executed directly on the real system's I/O device. DMKVSIEX calls DMKFREE to obtain free storage and then it constructs an IOBLOK in the storage obtained. The IOBLOK serves as an identifier of the I/O task to be performed. It contains a pointer to the channel program to be executed and the address of the routine that is to handle any interruptions associated with the operation.

DMKVSIEX stores the contents of the user's CAW in IOBCAW and sets the interruption return address (IOBIRA) to be the same as the virtual interruption return address (DMKVIOIN) in DMKVIO. The CCW translation routine (DMKCCWTR) is then called to locate and bring into real main storage all user pages associated with the channel program, including those containing data and CCWs. The following occurs:

- The CCWs are translated.
- A corresponding real channel program is constructed.
- The data pages are locked into real storage.
- DMKCCWTR returns control to DMKVSIEX. DMKVSIEX places the user in a pseudo wait state, IOWAIT, and calls the real I/O scheduler DMKIOSQV to schedule the I/O on the real configuration.

DMKIOSQV queues the request for operation on the real channel, control unit, and device corresponding to the address used by the virtual machine. When the real SIO is issued, DMKIOS takes the user out of IOWAIT and reflects the condition code for the SIO if it is zero. If it is not zero, the operation is further analyzed by DMKVIOIN. In any case, DMKIOSQV returns control to DMKVSIEX, which passes control to DMKDSPCH.

Other Privileged I/O Instructions

Other privileged I/O instructions are handled directly by DMKVSIEX. DMKVSIEX scans the virtual channel, control unit, and device blocks in the same manner as for a SIO and reflects the proper status and condition to the virtual machine. In some cases (TIO), the status of the addressed devices is altered after the status is presented.

If the operation active on the virtual device is actually in progress in the real equipment, the simulation of a HIO or HDV is somewhat more involved, since it requires the actual execution of the instruction. In this case, the active operation is halted and the resultant condition code/status is returned to the user.

Virtual Channel-to-Channel Support (CTCA and 3088)

Virtual channel-to-channel support simulates data transfer and control communication between two channel-to-channel devices, either on two distinct processors or two channels on a single processor. Data transfer is accomplished via synchronized complementary I/O commands (for example, READ/WRITE, WRITE/READ) issued to both parts of the CTC device. Each part of the CTC device is identical and the operation of the unit is completely symmetrical.

The VM/SP control program support for virtual CTC devices (channel-to-channel adapter and 3088) includes all status data, sense data, and interrupt logic necessary to simulate the operation of the real CTC device. Data transfer, command byte exchange, sense data, and status data presentation for the virtual CTC device is accomplished via storage-to-storage operations (MVCL, etc.). No real I/O operations (excluding paging I/O) or I/O interrupts are involved. Unit errors or control errors cannot occur.

Virtual Selector Channel I/O Requests

The CCW translator, DMKCCWTR, is called by the virtual machine I/O executive program (DMKVSIEX) when an I/O task block has been created and a list of virtual CCWs associated with a user's SIO request must be translated into real CCWs.

When the I/O operation from a self-modifying channel program is completed, DMKUNTIS is called by DMKIOS. When retranslation of OS ISAM CCWs is required, the self-modifying channel program checking portion of DMKCCWTR calls DMKISMTR.

DMKCCWTR operates in two phases:

- A scan and a translate phase
- A TIC-scan phase.

A self-modifying channel program checking function is also included.

The scan and translate phase analyzes the virtual CCW list. Some channel commands require additional doublewords for control information (for example, seek addresses). Additional control words are also allocated (in pairs) if the data area specified by a virtual CCW crosses 4096-byte page boundaries, or if the virtual CCW includes an IDA (indirect data address) flag.

Space is obtained from DMKFREE for the real CCW list, and the translation phase then translates the virtual CCW list into a real CCW list. TIC commands that cannot be immediately translated are flagged for later processing by the TIC-scan phase. A READ or WRITE command that specifies that data cross 4096-byte boundaries is revised to include an IDA flag that points to an indirect data address list (IDAL) and a pair of words for each 4096-byte page, in which each word handles a data transfer of 2048 bytes (or less). The real CCW is flagged as having a CP-generated IDA. DMKPTRAN is called (via the TRANS macro) to lock each 4096-byte page.

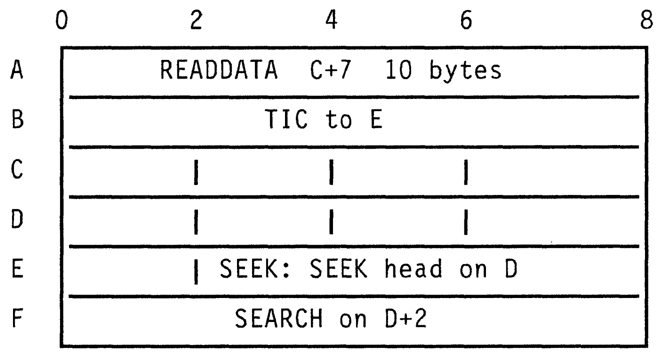
If the real CCW string does not fit in the allocated free storage block, a new block is obtained. The old block is transferred and adjusted before being released. The translation continues with the new block. The process is repeated, as needed, to contain the real CCW string.

Virtual CCWs having an IDA flag set are converted to user translated addresses for each IDAW (indirect data address word) in the virtual IDAL. DMKPTRAN is called for each IDAW. The CCW is flagged as having a user (but not CP) generated IDA.

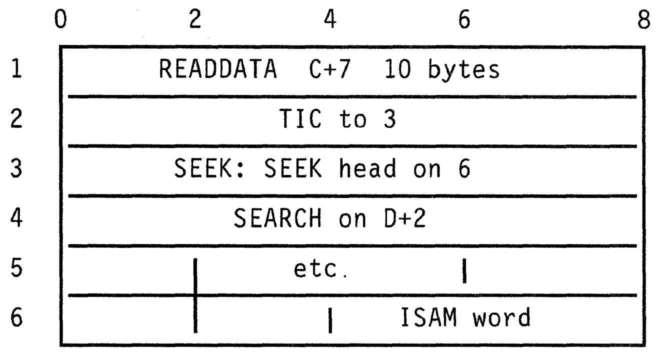
The TIC-scan phase scans the real CCW list for flagged (untranslated) TIC commands and creates a new virtual CCW list for the untranslated commands. Scan-translate phase processing is then repeated. When all virtual CCWs are translated, the virtual CAW in the IOBLOK task block is replaced by the real CAW (that is, a pointer to the real CCW list created by DMKCCWTR), and DMKCCWTR returns control to DMKVIOEX. The user protection key is saved.

OS ISAM Handling by DMKISMTR

Because many of the OS PCP, MFT, and MVT ISAM channel programs are self-modifying, special handling is required by the VM/SP control program to allow virtual machines to use this access method. The particular CCWs that require special handling have the following general format:



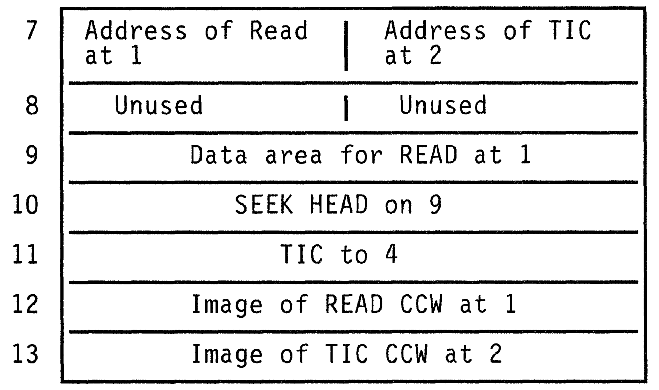
The CCW at A reads 10 bytes of data. The tenth byte forms the command code of the CCW at E. In addition, the data read in makes up the seek and search arguments for the CCWs at E and F. After the CCW string is translated by the VM/SP control program, it usually is in the following format:



To accomplish an efficient and non-timing-dependent translated operation for OS ISAM, the virtual CCW string is modified in the following manner.

DMKISMTR is called by DMKCCWTR if, during normal translation, a CCW of the type at 1 is encountered. The scan program locates the TIC at 2 by searching the translated CCW strings. The TIC at 2 locates the SEEK at 3.

The virtual address of the virtual SEEK CCW at E is located from the RCWTASK header. Seven doublewords of free storage are obtained and the address of the block is saved in the ISAM control word at 5. The seven doublewords are used to save the following information from the translated CCW strings:



The translated read CCW (at 1) is moved to the save block at 12. The TIC CCW (at 2) is moved to the save block at 13, and the addresses of 1 and 2 are saved at 7. The read CCW at 1 is modified to point to a 10-byte data area at 8+7 in the save block. The seek head CCW at 3 is copied into the save block at 10, and the seek address is modified to point to the data area at 9. At 11, a TIC CCW is built to rejoin the translated CCW string at 4. The search at 4 (or any subsequent search referencing D+2) is modified to point to 9+2. The completed CCW string has the following format:

```
 1   Readdata 8+7       10 Bytes
 2          TIC to 10
 3           Unused
 4       Search on 9 + 2
 5            Etc.
 6              |     ISAM  word
 7
 8       |     Unused      |
 9      Data Area for Readdata
10       | Seek Head on  9
11           TIC to 4
```

The interruption return address in the IOBLOK is set to DMKUNTIS. DMKUNTIS restores the CCWs to their original format from the seven doubleword extensions, moves the 10 bytes of data from 8+7 into virtual storage (at C+7), and releases the block. Normal I/O handling is resumed by DMKVIO and DMKUNT.

I/O Component States

The I/O components represented by the control blocks described in "Real I/O Control Blocks" on page 109 are in one of four states and the state is indicated by the flag bits in the block status byte. If the component is not disabled, it is either busy, scheduled, or available.

If the disabled bit is on, the component has been taken offline by the operator or the system and is at least temporarily unavailable. A request to use a disabled component causes the IOBLOK to be stacked with an indication of condition code 3 on the SIO and the real SIO is not performed.

An I/O unit is busy if it is transferring data (in the case of a channel or control unit), or if it is in physical motion (in the case of a device). If an I/O unit is busy, the IOBLOK for the request is queued from the control block representing that I/O unit.

An I/O unit is scheduled if it is not busy but will become busy after a higher-level component in the subchannel path becomes available and an operation is started. For example, if a request is made to read from a tape drive and the drive and control unit are available, but the channel is busy, the IOBLOK for that request is queued from the RCHBLOK for the busy channel and the RCUBLOK and RDEVBLOK of the drive and control unit are marked as scheduled. Future requests to that drive are queued from the RDEVBLOK for the scheduled device. When the channel completes the operation, the next pending operation is dequeued and started; the scheduled control unit and device are then marked as busy.

The IOBLOKs for various I/O requests indicate the status of that request by a combination of the status bits in the IOBLOK and the queue in which the block resides. In general, an IOBLOK is queued from the control block of the highest level I/O unit (taken from device up to channel) in the subchannel path is not available. Once the I/O operation is started, the IOBLOK is chained from the active IOBLOK pointer (RDEVAIOB) in the real device control block. Flags in the IOBLOK status fields may also indicate that a unit check has occurred, that a sense is in progress, or that a fatal I/O error (unrecoverable) has been recognized by error recovery procedures. After I/O control releases control of the IOBLOK, it is stacked on the queue of IOBLOKs and CPEXBLOKs anchored at DMKDSPRQ in the dispatcher and control is passed to the second-level interruption handler whose address is stored in IOBIRA.

I/O Interruptions

I/O interruptions are either synchronous or asynchronous. Asynchronous interruptions indicate the change in status of an I/O unit from the not-ready to ready state or busy to not-busy state. In either case, if the affected component has any pending requests queued from its control block, they are restarted, and whether or not the given interrupt is processed any further depends upon the status of the interrupting component. Channel-available and control-unit-end interruptions restart the interrupting component. An asynchronous device end is passed to the user if the device is dedicated; otherwise, the device is restarted.

An interruption is considered to be synchronous if the interrupting device has a nonzero pointer to an active IOBLOK. In this case, the following processing occurs:

- If a unit check has occurred, a sense is scheduled, and when the sense is completed, the appropriate ERP is called.
- If an ERP is currently in control of the task (indicated by a flag in the IOBLOK), return the IOBLOK to the appropriate ERP.
- If the operation is incomplete (for example, channel end is received without device end), the IOBLOK is copied and the copy is stacked but the original IOBLOK remains attached to RDEVAIOB to receive the final interrupt; then, the control unit and the channel is restarted.
- If the operation is complete (that is, the device is available), the IOBLOK is detached from the device and stacked, and the device, control unit and channel are restarted.

The restart operation usually dequeues the next IOBLOK that is queued to the restarted component and queues it to the next higher component in the subchannel path. When the channel level is reached, a SIO is issued and exit is taken to the dispatcher after handling any nonzero condition codes as previously described.

Virtual I/O Interruptions

When an I/O interruption is received, the IOBLOK is stacked for dispatching and control is passed to the address specified in the IOBIRA (interrupt return address) field. For operations requested by DMKVIOEX, the return address is DMKVIOIN (virtual interrupt return address). When DMKVIOIN receives control from the dispatcher, it loads the virtual address of the unit with which the interruption is associated from the IOBLOK and calls DMKSCNVU to locate the virtual device control blocks. DMKVIOIN then tests the IOBLOK status field to determine the cause for the interruption. If the block has been unstacked because of an interruption, the field is zero. If the operation was not started, it contains the condition code from the real SIO.

Note: The VIRA should not see a real condition code 2 as the result of a SIO, since channel-busy conditions are detected and reflected before any real I/O operation is attempted.

A condition code of 3 is reflected to the virtual machine and exit is taken to the dispatcher. For a condition code of 1, the CSW status field in the IOBLOK is examined to determine the cause for the CSW stored condition. The status is reflected to the virtual machine and various components of the virtual configuration may be freed, if the status so indicates. For example, if the CSW status indicated both channel end and device end, the operation was immediate and has completed. Thus, the CCW string (real) may be released and all virtual components marked available.

The CSW status returned for a virtual interruption must be tested in the same manner, with the additional requirement that the status be saved in the affected virtual I/O control blocks and that the CSW be saved in the VDEVCSW field for the device causing the interruption. If the unit check bit is on in the status field, the sense information saved in the associated IOERBLOK (pointed to by the IOBLOK) must be retained so that a sense initiated by the virtual machine receives the proper information.

In any case, when an interruption is received for a virtual device, a bit in the interruption mask, VCUDVINT, for the device's control unit is set to 1. The bit that is set is the one corresponding to the relative address of the interrupting device on the control unit. For example, if device 235 interrupts, the fifth bit in the VCUDVINT mask in the VCUBLOK for control unit 30 on channel 2 is flagged. Similarly, the bit in the VCHCUINT in the affected VCHBLOK is also set; in this case, bit 3 in VCHBLOK for channel 2. If the interruption is a channel class interrupt (PCI or CE), the address of the interrupting unit (235) is stored in the VCHCEDEV field in the VCHBLOK. The final interruption flag is set in the VMPEND field in the VMBLOK for the interrupted virtual machine; the bit set corresponds to the address of the interrupting channel. The next time, the virtual machine is dispatched and becomes enabled for I/O.

Monitoring I/O Activity

The missing interrupt handler, module DMKDID, receives control from a timer interrupt (TRQBLOK). The timer is reset, and DMKDID scans all of the real device blocks. If the RDEVBUZY or RDEVSCHD indicator flag is on, active I/O or device busy conditions exist. A check is made to make certain that the RDEVBUZY flag and the timer interrupt are for the same class of device. If the class is valid, the RDEVMID bit indicator flag is turned on. If the class is not valid, the scan continues.

RDEVMID on means that the device is active for the timer interval and a device interrupt is pending. When the device causes an interrupt, the indicator flags are reset by the first level interrupt handler, DMKIOT. If both flags (RDEVMID and RDEVBUZY) are on at the end of the *next* interval, a missing interrupt condition exists.

When a missing interrupt condition is detected, a CPEXBLOK is set up to give control to module DMKDID. The action taken depends on whether the request was queued or active. If the request was queued, a control unit end or device end is simulated. If the request was active, an interface control check is simulated.

If a virtual machine initiated the I/O operation, the interface control check is returned to the virtual machine. If CP started the I/O operation, the interface control check is handled by CP's ERPs.

Before either action is taken, a ten second timer is scheduled to return control to module DMKDID. When control is received from the timer, the RDEVMID flag is examined. If it is off (indicating that some I/O was completed), DMKDID sends a message to the system operator and a record is written to the error recording area indicating that the condition is corrected. If RDEVMID is on, the operator message and error record indicate the condition was not cleared.

Scheduling I/O Requests

A task that requests an I/O operation must specify the device on which the operation is to take place and must provide an IOBLOK that describes the operation. Upon entry to DMKIOS, register 10 must point to the IOBLOK. The IOBLOK must contain at least a pointer to the channel program to be started in IOBCAW and the address to which the dispatcher is to pass control in IOBIRA. In addition, the flags and status fields should be set to zero. If the operation is a VM/SP control program function such as for spooling or paging, the entry point DMKIOSQR is called. If the requester is the virtual I/O executive (DMKVIOEX) attempting to start a virtual machine operation, the entry point DMKIOSQV is called and some additional housekeeping is done. In either case, an attempt is made to find an available subchannel path from the device to its control unit and channel. If an I/O unit in the path is busy or scheduled, the IOBLOK for the request is queued to the control block of the I/O unit.

Requests are usually queued first-in-first-out (FIFO), except those requests:

- To fixed-head DASD preferred paging areas that are queued first
- To movable-head DASDs that are queued in order of seek address
- That release the affected component after initiation (SEEKS and other control commands) which are queued last-in-first-out (LIFO) from the control block.

Whether or not the operation has been successfully started, the caller requesting the I/O operation receives control from DMKIOS. If a free path to the device is found, the unit address is constructed and an SIO is issued. If the resulting condition code is zero, control is returned to the caller; otherwise, the code is stored in the requester's IOBLOK along with any pertinent CSW status, the IOBLOK is stacked, any components that become available are restarted, and control is returned to the caller.

In a multiprocessor environment, both processors have I/O capability. If either processor receives an I/O request, that processor attempts to initiate I/O operations.

At system generation time, when a channel path to a device is defined on one processor, an alternate logical path is automatically defined for the other processor. Thus, both processors *can* have access to any I/O device in the MP configuration.

If either processor receives an I/O request, that processor attempts to initiate the I/O operation on one of its own paths to the required device. If none of the online paths to the required device are available from the executing processor, that processor queues the I/O request on all busy and scheduled paths to the device, both its own and those of the other 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 for that processor.

While it is not required that both processors have access to all I/O devices, heavily used devices should be accessible by both processors to provide efficient system operation and to increase the possibility of system recovery following a processor or channel failure.

The I/O lock serializes access to the control blocks that represent I/O devices.

Alternate Path Scheduling

Alternate path I/O scheduling is performed according to the following scheme: DMKIOQ, called by DMKIOS, searches for an available path beginning with the primary path to the device. If an available path to the device exists, the I/O request is started immediately on the first available path to the device.

If the device is busy or scheduled, the IOBLOK is queued off the RDEVBLOK. No alternate path processing is performed at the device level.

If the device is not busy, not scheduled, nor offline, an 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 turned on. 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 before 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 byte multiplexor is marked busy is after a condition code 2 has been encountered. The I/O load on byte multiplexor channels must be sufficient to cause channel busy conditions before path selection on an alternate channel can take place.

If a busy or scheduled path is encountered, an IOBLOK is queued to the real block (RCUBLOK or RCHBLOK) and the search continues for an available path. If more than one busy path is encountered, multiple IOBLOKs are queued for the same I/O request. This is accomplished by creating mini IOBLOKs for each busy/scheduled path after the first. The primary IOBLOK is queued off the first busy path encountered. The mini IOBLOK is 16 bytes in length and consists of the first two doublewords of the IOBLOK, which is the same as the current IOBLOK structure. The IOBLOK and associated mini IOBLOKs are chained in a single-threaded queue by means of the IOBLINK field. The active IOBLOK pointer is not stored in the IOBLINK field until just prior to the SIO. Zeros are stored in IOBLINK at entry to DMKIOSQR to indicate no mini IOBLOKs have been queued as yet. See Figure 25 on page 121 for an example of mini IOBLOK queuing.

The last two words of the mini IOBLOK (IOBFPNT and IOBBPNT) are used as the double-threaded queue pointers for the RCUBLOK/RCHBLOK from which it is queued. A flag is set in the mini IOBLOK to identify it as a mini IOBLOK.

Figure 26 shows a sample control block structure when mini IOBLOKs are queued.

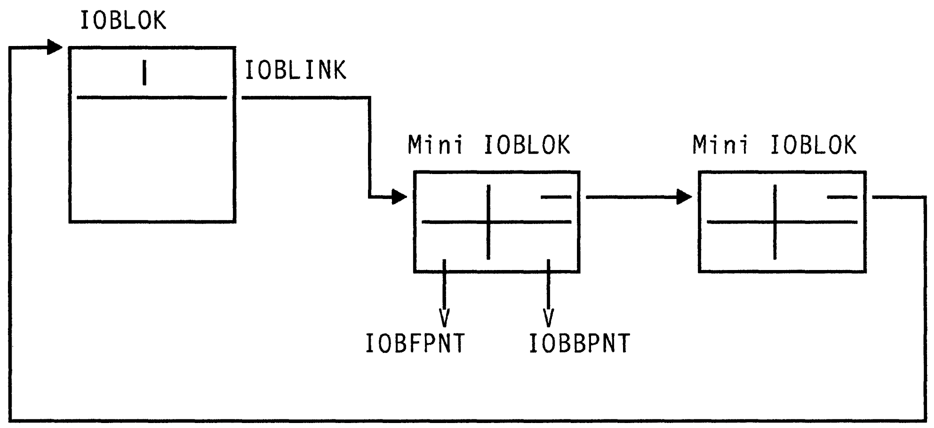


Figure 25. Mini IOBLOK Queuing

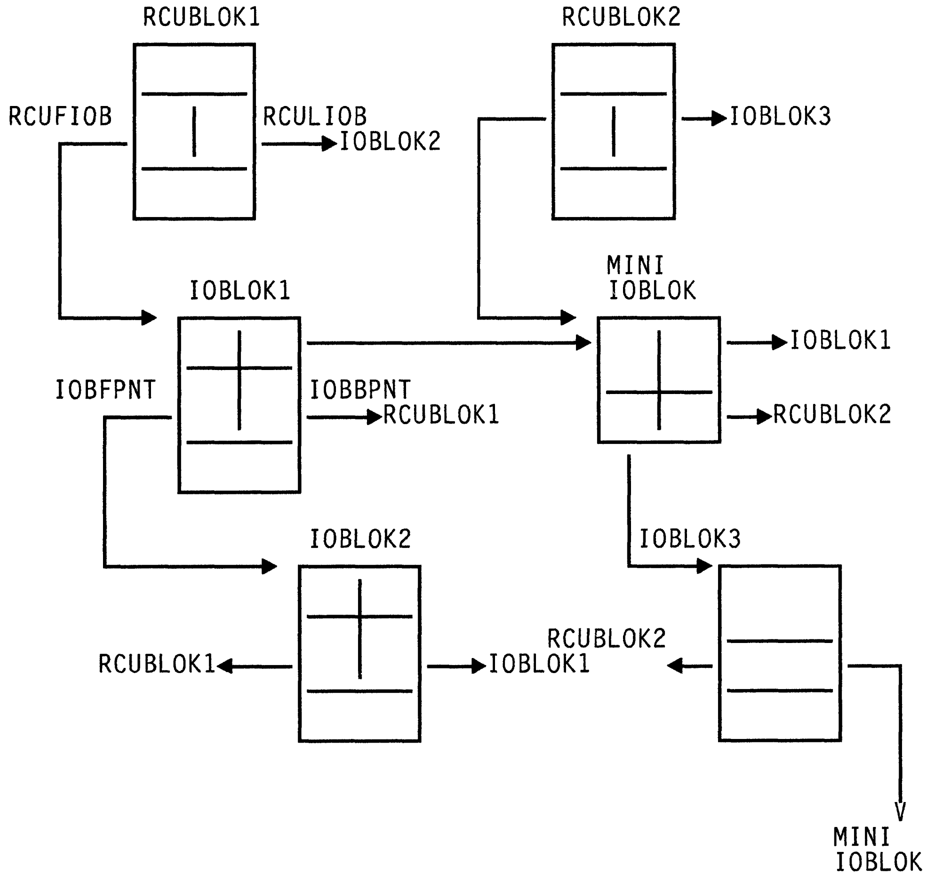


Figure 26. Control Block Structure for Alternate Path Request

Prior to starting an I/O operation associated with the request, a check is made to see if the IOBLOK is a mini IOBLOK and whether mini IOBLOKs are queued off this IOBLOK. All mini IOBLOKs associated with this request are dequeued from their respective queues by running the IOBLINK chain. The storage for the blocks is released. If the active IOBLOK is a mini IOBLOK, the IOBRADD from the mini IOBLOK is moved to the primary IOBLOK and the I/O started using the primary IOBLOK.

Reserve/Release

Reserve/release is supported for shared DASD as though each virtual machine has a separate channel path to a shared device. Reserve/release support prevents the occurrence of a channel lockout situation. This is accomplished by changing reserve CCWs to sense CCWs when a reserve is issued to a device that has alternate paths defined to it. In an MP configuration, alternate path is defined if there is a channel path from each processor. This means that whenever alternate paths are defined to a device, the real reserve does not execute on the hardware. Reserve/release support is implemented in VM/SP on a virtual basis allowing the reserve/release operation codes to be simulated on a virtual basis for minidisks, including full-extent minidisks. When a reserve is issued against a minidisk, the reserve is accomplished by a locking mechanism. The status of the minidisk is maintained in the VRRBLOK that is chained from the VDEVBLOK.

The following matrix identifies how the reserve operation code is handled in the various situations:

	Defined Alternate Paths to Device[1]	**Will Reserve/ Release Execute on the Hardware**	**Virtual Reserve/ Release Requested for Minidisks**	**RESERVE**[2] **--or-- SENSE**[3]
Dedicated DASD or Tape	NO	N/A	N/A	RESERVE
	YES	N/A	N/A	SENSE
Minidisk	NO	NO	NO	RESERVE
	NO	NO	YES	SENSE
	NO	YES	NO	RESERVE
	NO	YES	YES	RESERVE
	YES	N/A	N/A	SENSE

[1]On an MP system, alternate paths are defined if each processor has a channel path to the device.
[2]The RESERVE keyword in the chart indicates that the real reserve is allowed to execute on the hardware.
[3]The SENSE keyword indicates that the reserve CCW is changed to a sense CCW. Virtual Reserve/Release is requested by means of a new option on the MDISK directory control statement.

DMKVIO performs the following steps when virtual reserve/release processing is requested:

1. DMKVSI calls DMKCCW to perform CCW translation. For DASD devices, DMKCCW checks if the virtual reserve/release feature bit is on in the VDEVBLOK. If virtual reserve/release processing has been requested and if the device is not reserved by anyone or it is reserved by this user, processing continues normally. If the device is reserved by another user, DMKCCW calls DMKUNTFR to restore the CCWs to their original state and returns to the caller, unless sense bytes have been transferred to the user's storage in which case CP enqueues on the minidisk and waits until it is no longer reserved at which time the I/O can proceed. If the I/O request can continue and the CCW chain contains a reserve command, the VDEVBLOK and the VRRBLOK are flagged as reserved. If the CCW chain also contains a release, the IOBLOK is flagged to indicate to DMKUNTFR to release the virtual disk. Control returns to DMKVSI.

2. DMKVSI reflects a device-busy condition to the virtual machine if the minidisk is currently reserved by another user.

3. DMKUNT reflects a device end interrupt to all virtual machine users who previously received a busy condition, when the device is released.

Ordered Seek Queuing: Requests to start I/O on system devices are normally handled first-in-first-out. However, requests to movable-head DASD devices are queued on the device in ascending order by seek address. This ordered seek queuing is performed to minimize intercylinder seek times and to improve the overall throughput of the I/O system.

CP assumes that very few virtual machines perform chained SEEKs. Therefore, the first logical address represents the position of the arm upon completion of the I/O operation. Ordered SEEK queuing is based on the relocated real cylinder. DMKIOQ, which handles IOBLOK queuing, uses the cylinder location supplied in IOBCYL for ordered seek queuing. This field is initialized by the calling CP routine for paging and spooling or by the CCW translator for virtual I/O. The CCW translator, DMKCCW, supplies the IOBCYL value in the following manner:

- Reads the IPL record, relocates to virtual cylinder 0
- Recalibrates, issues a real calibrate, and then a SEEK to virtual cylinder 0
- Issues a channel SEEK, relocates to the virtual cylinder
- For FB-512 devices, converts the block number being located to the corresponding cylinder number.

The IOBLOK queuing subroutine of DMKIOQ recognizes that a request is being queued on a movable-head DASD by means of the device class and type fields of RDEVBLOK. Instead of adding the IOBLOK to the end of the queue on the RDEVBLOK, the queuing routine sorts the block into the queue based on the cylinder number for the request. The cylinder number for any request to DASD is recorded in the IOBCYL field. The queue of IOBLOKs on a real device block is sorted in ascending order by SEEK address, unless the entire device is dedicated to a given user. In this case, DMKIOQ does not automatically schedule the device, and no more than one request can be outstanding at any one time.

When an outstanding I/O request for a device has completed, DMKIOS attempts to restart the device by dequeuing and starting the next IOBLOK queued on the device. For non-DASD, this is the first IOBLOK queued. Fixed-head DASD paging requests are dequeued ahead of moveable-head requests. For movable-head DASD, the queued requests are dequeued in either ascending or descending order, depending upon the current position (recorded in RDEVCYL) and the direction of motion of the arm. If the arm is seeking up (that is, toward the higher cylinder numbers), the queue of IOBLOKs is scanned from the first block toward the last until an IOBLOK is found with an IOBCYL value equal to or greater than the value in RDEVCYL, or until the end of the queue is reached. At this point, the device is flagged as seeking down and the queue is scanned from last to first until an IOBLOK with an IOBCYL value equal to or less than RDEVCYL is found. When the IOBLOK is found, it is dequeued and started. The direction of motion is indicated by an RDEVFLAG bit and the next request is dequeued downward until the head of the queue is reached.

Because the queue itself is a two-way chained list, no special handling for null or unity set lists is required, and the ordered seek algorithm returns to first-in-first-out queuing.

Dedicated Channel Support: One of the facilities of the VM/SP control program allows a virtual machine to control one or more channels on a dedicated basis. The channels are attached to the virtual machine by using the privileged ATTACH CHANNEL command. A virtual machine can have one or more dedicated channels. In addition, channels can be split between virtual machines but a dedicated channel cannot be shared between two virtual machines. For instance, channel 1 could be dedicated to virtual machine A, and channel 2 could be dedicated to virtual machine B, or both could be dedicated to virtual machine A or B.

With a dedicated channel, all virtual machine device addresses must be identical to the real machine device addresses. For instance, virtual device 130 must be real device 130, and virtual device 132 must be real device 132. With dedicated channels, CP does not perform any virtual device address mapping.

CP error recording and channel recovery procedures are still in effect for dedicated channels. The dedicated channel support can be used in conjunction with the virtual=real feature for any virtual machine that is occupying the virtual=real storage space.

Virtual Console Simulation

DMKVCN receives control from the virtual machine I/O executive, DMKVSI. When control is received, the device is available with no interruptions pending. A console control block, VCONCTL, that is obtained from storage and chained from the virtual device control block, VDEVBLOK, by DMKLOG is accessed for use during the interpretation of the virtual console I/O sequence. The user's CAW is examined for validity. If it is valid, the TRANS macro is issued to fetch the first user CCW. This CCW is moved to the VCONCTL block for analysis.

The CCW is analyzed to determine if it is a read, a write, a control, a sense, a TIC, or an invalid operation. Based upon the analysis, the appropriate processing routine in DMKVCN is invoked.

The Read Simulation Routine: Obtains a buffer for input data from free storage. The location of the buffer is set in the VCONCTL block. The DMKQCNRD routine is called to schedule and perform an actual read to the corresponding real device representing the user's virtual console. If SET LINEDIT ON is specified, the buffer data is edited and translated to EBCDIC. When the read is completed, the data is moved to the specified user address obtained from the address portion of the virtual CCW. If command chaining is specified, processing returns to fetch and analyze the next CCW. If command chaining is not specified, the virtual CSW is constructed in the VDEVBLOK and an interrupt is flagged as pending in the VMBLOK.

The Write Simulation Routine: Obtains a buffer for the construction of the output message from free storage. The virtual machine data is located from the virtual CCW address in the VCONCTL block and moved to the data buffer. The DMKQCNWT routine is called to write the data in the buffer and provide the necessary length, translation, and format functions. Control is received at the DMKVCN module upon completion of the writing. At this point, the virtual CCW is re-examined. If command chaining is specified, processing continues to fetch and analyze the next CCW. If command chaining is not specified, the virtual CSW is constructed in the VDEVBLOK and an interruption is flagged as pending in the VMBLOK.

The Control Simulation Routine: Is used for the NOP and ALARM operations. A NOP operation requires no data transfer or I/O operation. An ALARM operation has no equivalent on low-speed teleprocessing equipment; thus, a message indicating the ALARM operation is constructed. DMKQCNWT is called to output the constructed message. If the command is chained, processing continues (for NOP or ALARM) to fetch the next CCW and analyze it. If command chaining is not specified and this is not the first CCW, a virtual CSW is constructed in the VDEVBLOK and an interruption is flagged as pending in the VMBLOK. If this is the first (and only) CCW, then a condition code of 1 is presented with channel end and device end in the virtual CSW.

A Virtual Sense Operation: Is similar to a control operation, because no actual I/O operation is performed. However, there is data transfer. The sense data from the VDEVBLOK is moved to the virtual storage location specified in the virtual CCW address. If the command is chained, processing continues to fetch the next CCW and analyze it. Otherwise, an interruption is flagged as pending in the VMBLOK.

A Virtual TIC Operation: Fetches the virtual CCW addressed by the TIC address and analyzes the fetched CCW. If the fetched CCW is itself a TIC, or if the TIC is the first CCW, a channel program check condition is reflected to the virtual machine as an interruption or as a CSW-stored condition, respectively.

Invalid Operation: Any other operation is considered invalid. Command reject status is posted in the virtual sense byte and the operation is terminated with unit check status presented in the virtual CSW.

Remote 3270 Programming

For a basic understanding of CP processing of data relating to 3270 devices on binary synchronous lines, the information and terminology contained in *IBM 3270 Information Display System Component Description*, and *IBM General Information - Binary Synchronous Communications* is required.

A digest of some of this essential information as it applies to VM/SP follows:

- Text messages to and from remote terminals and printers can only be achieved when the bisync line is in text mode.
- Text messages from a remote device can be the result of a general poll or specific poll operation to the related device or devices on the bisync line. This polling communication interface is accomplished by each line-connected control unit having unique specific poll and general poll recognition circuitry and by the CP terminal list of valid bisync lines and 3270 remote control unit addresses. This list, the terminal list, is generated by VM/SP system generation procedures employing TERMINAL and CLUSTER macros. For more details, see the *VM/SP Planning Guide and Reference.*
- Reliability and dependability of line operation is achieved by the use of: a double addressing scheme, control characters with a rigid message protocol, and complex redundancy-check characters appended to transmission messages. Examples of these techniques are shown in the formats that follow.
- Every message (text or control) that is issued by CP may or may not be responded to by the remote station or control unit. The type of response (or absence of response) that CP receives depends on the receptiveness of that device or control unit to the previously sent message (is the device ready and enabled and accurately addressed) and the content and correctness of the message (no line errors).

- To establish the relationship of the line of terminal response to a particular line or device write or read operation, CP employs an operation `tracking` facility (TP op code) imbedded in the issued CCWs. The function performed by the CP op code is described in the following CCW formats.

Format of the 3270 Remote CCW

Operation Code 1 byte	Address Field 3 bytes	Flags 1 byte	TP Op Code 1 byte	Count 2 bytes
0 7	8 31	32 39	40 47	48 63

where:

Operation Code
: contains the hexadecimal value of the type of operation performed by the command.

Valid operation codes are:

X'01'	WRITE
X'02'	READ
X'03'	NOOP
X'09'	POLL
X'23'	SET MODE
X'27'	ENABLE
X'2F'	DISABLE

Address Field
: Depending on CCW usage, this field may address:

Data
: Output data contained in the CONTASK

Area
: The address of the data area (read buffer) located in the BSCBLOK at BSCREAD.

Control Characters
: A data-link control character such as EOT or ENQ that is defined in DMKRGA or DMKRGB.

Response
: (BSCRESP). The address location of the response buffer in the BSCBLOK.

Addressing Characters
: Data characters that indicate which device is to send or receive data. The entry for WRITE operation is at location BSCSEL. The entry for the READ operation is at location BSCPOLL.

Note: To see how the key words DATA, Control Characters, Response and addressing Characters are used, refer to the CCW sequences described in "I/O Programs for Bisynchronous Lines and Remote 3270s."

Flags
: The flag bits turned on in the CCW: CC (channel commands), CD (chained data), SILI (suppress incorrect length indication), skip (suppress data transfer to main storage).

TP Op Code
: An imbedded teleprocessing operation code in the CCWs used in bisync line communications. This code is inspected by the secondary interruption handler, DMKRGAIN, when channel end and device end are received. The code is also used by the error processing module, DMKBSC. The code indicates the function being performed by he associated command. For use of the TP op codes, refer to the formatted CCWs that follow.

Count
: Refers to the byte length of the CCW READ or WRITE operation.

I/O Programs for Bisynchronous Lines and Remote 3270s

Before data communication to remote 3270 equipment can take place, the remote teleprocessing line, the control unit and the device(s) must be enabled for communication. This occurs when a special sequence of channel commands is executed. Disabling a line occurs in a similar manner. The following is the format of the CCWs used in the enabling/disabling operation:

Enable a Line

Operation	Command Code	Address	Flags	TP Op Code	Count
Disable Line	X'2F'	0	CC, SILI	01	1
Set Mode	X'23'	Control Character	CC, SILI	01	1
Enable Line	X'27'	0	SILI	01	1

Disable a Line

Operation	Command Code	Address	Flags	TP Op Code	Count
Disable Line	X'2F'	0	SILI	01	1

After a line is enabled, communication can then be directed to a particular resource. The sequence of events is as follows:

Send a data link control character on the line that places the control unit in control mode. This mode makes the control unit receptive to the specific address indicated by the second CCW. The third CCW is a read CCW that is needed for the acknowledgement response from the addressed control unit. Normally, in response, CP transmits a block of data to that device with a write text CCW. Acknowledgement of receipt of this data is contained by the read response (text) CCW. The format of the CCW select and write text operation follows.

Write Select

Operation	Command Code	Address	Flags	TP Op Code	Count
Write an EOT	01	Control Character	CC, SILI	02	1
Write addressing character	01	List Address Character	CC, SILI	03	5
Read Response	02	Response	SILI	05	2

When the control unit recognizes the addressing characters, it transmits an acknowledgement and enters text mode. CP then writes one or more screen lines in a Write Text operation. Multiple lines are written if more than one CONTASK is queued or if the screen must be updated.

Write Text (one line)

Operation	Command Code	Address	Flags	TP Op Code	Count
Write text	01	Data	CD, SILI	03	variable
Write ETX	01	Control Character	CC, SILI	03	1
Read Response	02	Response	SILI	11	2

Write Text (multiple lines) - part 1

Operation	Command Code	Address	Flags	TP Op Code	Count
Write text	01	Data	CD, SILI	03	variable
TIC	08	To next write (format (1)or(2))	CD, SILI	03	1

Write text (multiple lines) - part 2

Operation	Command Code	Address	Flags	Tp Op Code	Count
Write text	01	Data	CD, SILI	03	variable
Write ETX	01	Data	CC, SILI	03	1
Read Response	02	Response	SILI	11	2

In situations where the line is found to be in text mode, CP can issue a write reset sequence to put the binary synchronous line in control mode. The following format illustrates the write reset CCW.

Write Reset

Operation	Command Code	Address	Flags	TP Op Code	Count
Write EOT	01	Control Character	SILI	09	1

In situations where the expected response from a remote station was not received or was invalid, the channel program may request the remote station to retransmit the response. The following write ENQ format shows this sequence. The remote station, upon receipt of the ENQ message, responds by transmitting the expected or valid response to the response area indicated by the original read response CCW.

Write ENQ

Operation	Command Code	Address	Flags	TP Op Code	Count
Write ENQ	01		CC, SILI	06	1
TIC	02	Original Read Response CCW	SILI	0	1

Read operations occur following a general poll or a specific poll for text messages. In a general poll sequence, CP transmits the general poll characters to the attached control unit on the bisync line. The control unit recognizes the polling request, and transmits any pending data, causing the second (NOP) CCW to be skipped. The last CCW provides the read buffer and the count necessary for the incoming data block from the first remote station on the list that had a message queued for transmission. If, however, there is no pending data, then the channel program ends with the third CCW. The following read initial format shows the initial read CCW sequence.

Read Initial

Operation	Command Code	Address	Flags	TP Op Code	Count
Write EOT	01	Control Character	CC, SILI	02	1
Poll	09	Control Character	CC, SILI	03	7
I/O No-operation	03	0	SILI	07	1
Read Text	02	Area	SILI	10	263

After CP receives a message from a remote station, it transmits any outbound data then repeats the general poll. When a poll completes with no inbound data, CP starts the poll delay, then allows the poll delay interval to expire before starting another poll to the line (assuming CP has no higher line priority tasks to process). If, in the process of receiving messages from remote stations, CP receives a message block that is invalid or its beginning or ending bisync control characters are not recognized, CP can elect to send a negative response back to the remote station. This negative response, the NAK control character, causes the remote station to retransmit the previous message to CP; this incoming message is processed by the second CCW of the read repeat sequence as shown in the format below.

Read Repeat

Operation	Command Code	Address	Flags	TP Op Code	Count
Write NAK	01	Control Character	CC, SILI	06	1
TIC	08	Original Read Text CCW	SILI	00	1

Once an error-free message is received from the remote station, CP proceeds with one of two actions:

- If there is data to be transmitted, CP sends it immediately, using the Write Text format. This is known as binary synchronous `limited conversational mode.`

- If there is no data to be transmitted to the remote device that generated the message, CP sends an RVI control character to suspend the transmission of additional data. The remote 3270 responds to the RVI with EOT instead of sending another message.

Read Interruption

Operation	Command Code	Address	Flags	TP Op Code	Count
Write RVI	01	Control Character	CC, SILI	06	2
Read Response	02	Response	SILI	11	2

Data Formats - Remote 3270s

CP, in conjunction with remote 3270 support, uses the following formats for its text messages. For a detailed explanation of the abbreviations used, see the *IBM 3270 Information Display System Component Description.*

Write Text Data Message Format

Display commands use this message format for the placement or erasure of data anywhere on the display screen. The display commands that implement this function are: WRITE (X'F1'), ERASE/WRITE (X'F5'), ERASE WRITE ALTERNATE (X'7E'), and COPY (X'F7').

Write Data Stream

STX	ESC	CMD	WCC	SBA	Buffer Address	Orders & Text	SBA	Buffer Address	//	ETX
1	1	1	1	1	2	variable	1	2		1

Write Structured Field Data Stream

CP uses the Write Structured Field data stream when the display is enabled by the system operator using the READ PARTITION (QUERY) function. CP determines the device features of 3278 and 3279 display stations. This data stream is not used if the display station type is 3277.

DLE	STX	ESC	WSF CMD	length	RD. PART.	X'FF'	QUERY
1	1	1	1	2	1	1	1

If the remote control unit supports the 3270 extended data stream, then CP transmits all text messages in binary synchronous *transparent text* mode. Transparent text mode is used so that all 256 possible character codes may be transmitted. These character codes are required for certain 3270 extended functions such as, color and extended highlight features.

The format of the Write Data Stream in transparent text mode is similar to the normal format except that the leading STX and trailing ETX are modified:

DLE	STX	ESC	CMD	WCC	SBA	Buffer Address	Orders & Text	SBA	// Buffer Address //	DLE	ETX
1	1	1	1	1	1	2	variable	1	2	1	1

The DLE-STX header indicates the start of a transparent text block; the trailing DLE-ETX indicates the end of the block. The DLE-ETX is written by a special *command-chained* CCW to differentiate it from the data contained in the text.

The transmission control unit (TCU) hardware provides special support for transparent text mode. The TCU checks each text character and inserts a DLE character before any DLE text character, thus turning a single DLE into a double DLE. These DLEs are automatically removed by the remote control unit. See *IBM General Information - Binary Synchronous Communications* for details.

Write Text Messages for the Copy Command

The COPY command is limited to compatible printers located on the same control unit. Action starts by pressing a PF key designated for the COPY function. CP responds by sending a message to the control unit that contains both the designated printer and the display station that requested the action and directs the control unit to print the designated display buffer to the printer specified.

The format of the COPY messages follows:

3271 Copy Data Stream

STX	ESC	CMD X'F7'	CCC	From Address	ETX

3275 Copy Data Stream

STX	ESC	CMD X'F1'	WCC	SBA	Buff Adr (4040	IC	ETX

Read Text and Read Header Message Formats

The following is representative of input message formats. The format of a CP-generated read operation follows.

Read Text Data Stream

Index Byte	STX	CU Adr	Dev Adr	AID	Cursor Addr	SBA	Buff Addr	Text	SBA	Buff Addr	Text		ETX

The Read Text Data Stream is in the *transparent text* format if there are binary synchronous control codes in the data.

Index Byte	DLE	STX	CU Adr	Dev Adr	AID	Curs Addr	SBA	Buff Adr	Text	SBA	Buff Addr	Text	DLE	ETX

Error Status Data Stream

Another form of input message is the error status message. Error status is processed by the DMKRGA module. The characters, %R, following the SOH signify that this message contains sense and status data. The format of this message follows.

Index Byte	SOH	%	R	STX	CU ADR	Dev Adr	Sense/ Status Bytes	ETX

Test Request Data Stream

The test request message, upon receipt from display terminals, is ignored by CP. The input inhibit mode that the display terminal enters upon pressing the test request key can be reset only if the terminal user presses the RESET key. The characters, %/, following SOH indicate the test request function. The format of this message follows.

Index Byte	SOH	%	/	STX	Text	ETX

Allocation Management

Real storage space above the Control Program nucleus is made up of the dynamic paging area and the free storage area. Page frames (allocation space in real storage for a page of data) in the dynamic paging area are allocated to virtual machines and the control program to satisfy paging requests. Blocks of storage, requested by virtual machines and CP for working storage, are allocated from the free storage area.

Normal Paging Requests

If a program interruption is caused by a normal paging request (not from a virtual machine that is running in EC mode with translation on), DMKPRGIN determines whether a segment or page translation error has occurred. If one of these errors occurred, an invalid address interruption code is set, and the interruption is reflected to the virtual machine supervisor. If a segment or page translation error has not occurred, the virtual machine's current PSW is updated from the program old PSW (PROPSW), the address of the current VMBLOK is placed in register 11, and DMKPTRAN is called to obtain the required page. When the paging operation is completed, control is returned to DMKDSPCH.

Virtual Storage Management

When operating in the CP relocate environment, each virtual machine's virtual storage space is described by two sets of tables.

- One set, the segment and page tables, describes the location and availability of any of the virtual machine's virtual pages that may be resident in real storage. Locations in these tables are indexable by virtual address, and the entries contain index values that reference corresponding real storage addresses. In addition, each table entry contains an indication of whether the corresponding virtual page is available to the user in real storage. These tables are referenced directly by the DAT feature when the virtual machine's program is running.

- The second set of tables, called swap tables, is a map of the locations of the virtual machine's pages on the DASD devices that comprise the system's paging or auxiliary storage. The DASD addresses in these tables can either represent the source of a page of virtual storage (the location to which a page may be moved if necessary) or a dummy address, indicating that the given page has not yet been referenced and, thus, has a value of binary zeros.

  The swap tables are arranged in a format indexable by virtual storage address. In addition to containing the address of a page, each entry contains flags and status bytes that indicate such information as:

  – The storage protection keys to be assigned to the page when it is made resident

  – Whether the page is currently on its way into or out of the system (in transit), etc.

These tables are not referenced directly by the hardware as are the page and segment tables, but are used by paging management to locate user pages that are needed to execute a program.

Virtual storage management is done by the technique known as demand paging. This means that a page of virtual storage is not paged in from its DASD auxiliary storage area until it is needed. CP does not determine the pages required by a virtual machine before the virtual machine executes. A demand for a page can be made either implicitly by the virtual machine or explicitly by CP.

- An implicit demand for a page is made when a program attempts to reference a page that is not available in real main storage. This attempt causes a program interruption with the interruption code indicating a page or segment exception. Upon recognition of this condition, control is passed to the paging manager to obtain a page frame of real main storage and to bring in the desired page.

- An explicit demand for a page can be made by CP (for example, in the course of translating a user's channel program). If, in the process of translation, CP encounters a CCW that addresses a page that is not resident in real storage, a call is made to the paging manager to make the referenced page resident.

While the requested page is being fetched, the requesting virtual machine is unable to continue execution; however, it may be possible to run other tasks in the system, and CP runs these while the needed page is being paged in. When the requested page is resident, the virtual machine can be run and is dispatched in its turn.

In addition to demanding pages, virtual machines implicitly or explicitly release page frames of their virtual storage space. Part of the space may be explicitly released from both real and virtual storage via a DIAGNOSE instruction that indicates to the control program those page frames that are to be released. An entire virtual storage is released when a user loads (via IPL) a new operating system or logs off from the system.

CP also has virtual storage associated with it. This space contains CP (some parts of which need not always be resident in real storage), and virtual storage buffers for spooling and system directory operations. Although CP makes use of virtual storage space for its execution, it does not run in relocate mode. Thus, nonresident modules must be completely relocatable.

To improve performance by eliminating unnecessary LCTL and LRA instructions, CP keeps track of where each virtual machine's page zero resides. CP does this by checking 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 it is not resident, CP issues a TRANS macro, which checks for page residency and demands a page-in if the page is not in real storage. If the page is resident, CP bypasses issuing the TRANS macro.

Virtual Buffer Manager

CP can dynamically increase the amount of virtual storage it can use by using the virtual buffer manager. This virtual buffer manager also allows CP to allocate and deallocate this virtual storage.

The virtual buffer manager maintains a chain of nondispatchable VMBLOKs, similar to the SYSTEM VMBLOK. CP modules can invoke the virtual buffer manager to create these VMBLOKs dynamically. These VMBLOKs, which have userids defined by the calling CP module, are maintained on a chain that is anchored off the SYSTEM VMBLOK and not associated with the user VMBLOK chain.

As with the SYSTEM VMBLOK, each of these VMBLOKs is associated with segment, page, and swap tables for virtual storage. CP maintains an allocation table so it can control the allocation and deallocation of the virtual storage associated with the VMBLOK. The system function using the virtual buffer manager must be sensitive to the VMBLOK associated with the virtual storage it uses: paging operations must be done using the specific VMBLOK associated with the virtual storage obtained to get access to the correct segment, page, and swap tables needed to access that storage.

Real Storage Management

Real storage management allocates the system's page frames of real storage to satisfy the demands for virtual pages made by the system's virtual machines. Efficiency of allocation involves a trade-off; the paging manager uses only enough processor time to ensure that:

- The set of virtual storage pages that are resident represent those pages that are most likely to be used.
- A sufficient number of cycles is available to execute virtual machine programs.

Inefficiency in the first area causes a condition known as thrashing, which means that frequently used pages are not allowed to remain resident long enough for useful work to be performed by or on them. Thrashing could be aggravated by the paging manager's page frame selection algorithm or by a dispatcher that attempts to run more tasks than the system can handle (the sum of their storage requirements exceeds the real paging space available in the system). Thus, the paging manager must keep statistics on system and virtual machine paging activity and make these statistics available to the dispatcher to detect and prevent a potential thrashing condition.

Inefficiency in the second area causes an unacceptable ratio of CP overhead to virtual machine program time, and in extreme cases may cause CP to use excessive processor time. To understand how allocation is determined by CP, the way in which the inventory of real storage page frames is described to the system must be understood.

Each page frame (4096-byte block) of real storage in the system is in one of two basic states: non-pageable or pageable. A nonpageable page must remain resident in real storage for some period of time; thus, the page frame cannot be taken from its current owner to be given to someone else. Pages can be either permanently or temporarily non-pageable, depending upon their use.

Temporary locks usually occur when an I/O operation has been initiated that is moving data either to or from the page, and the page must be kept in real storage until the operation has completed.

A page can also be temporarily non-pageable if it contains an active nonresident CP routine.

In addition, a page can be non-pageable through use of the LOCK command. Pages locked this way are permanently resident until they are explicitly unlocked by the UNLOCK command. Pages that are usually considered permanently non-pageable are those that contain the resident portion of CP and those that contain the system's free storage area in which control blocks, I/O buffers, etc., are built.

The data area that page management routines use to control and allocate real storage is the CORTABLE. Each page frame of real storage has a corresponding entry in the CORTABLE, and because the table entries are fixed in length and contiguous, the entry for any given real page frame may be located directly by indexing into the table. Each entry contains pointers that indicate both the status and ownership of the real page that it represents. Some pointers link page table and swap table entries to the real page (and thus establish ownership), while others link the entry into one of several lists that the paging routines use to indicate the page frame's status and availability for paging. A given CORTABLE entry may appear on either of two lists if its real page frame is available for paging; however, if the page referenced is locked or is in transit, its entry is not in either list and is not referenced when available page frames are being searched for swap candidates. The lists are known as the free list (FREELIST) and the flush list (FLUSHLST), and they represent various levels of page frame availability.

The free list contains page frames that are immediately available for assignment to a requesting virtual machine. The virtual storage pages for which they were last used have either been released by their owners or they have been paged out to auxiliary storage. Requests for real storage are always satisfied from the free list. If the list has been depleted, the requestor waits until a new page frame becomes available as the result of a virtual storage release or a swap-out.

The flush list contains page frames that belong to those virtual machines that have been dropped from an active dispatching queue. The flush list is the first place that the page frame selection routine looks to find a page to swap out or to assign to the free list for a virtual machine that requires real storage space.

Requests for Real Storage Page Frames

Requests for real storage fall into two general categories: those that are requesting space for a page of virtual storage, and those (such as requests for CP work space) that need page frames for their own use. The former, more general case is discussed first, because the latter case is a subset of the first.

The main page manager routine, DMKPTRAN, maps a request for a specific virtual storage address into a page frame of real storage. This requires that the virtual page be read in and the necessary tables be updated to show the proper status of the page frame.

DMKPTRAN requires that the caller supply only the virtual address to be translated and any options that apply to the page to be located. Most calls are made via the TRANS macro, which sets up the necessary parameters, determines whether or not the required page is resident, and calls DMKPTRAN if it is not.

When DMKPTRAN receives control, it first tests to see if the requested page is resident. This is done via the LRA instruction. If the page is resident, the routine locks the page if requested and exits to the caller. If the LRA indicates that the page is unavailable, it is still possible that the required page is resident. This occurs if the page frame has been placed on the FREELIST but has not been assigned to another virtual machine. When the page swap routine removes a page frame from a virtual machine, the unavailable bit is set in the corresponding page table entry; however, the real main storage index for the page frame is left unchanged. The page table entry is set to zero only when the corresponding page is actually assigned to another virtual machine. Thus, if DMKPTRAN finds the page unavailable, a further test is made on the page table entry to see if the page can be reclaimed. If the entry is not zero (aside from the unavailable bit), the CORTABLE entry for the page frame is removed from the FREELIST and the page frame is returned to the calling virtual machine.

If the page table entry corresponding to the requested virtual page is zero, the required page is not in real storage and must be paged in. However, it is possible that the page is already on its way into main storage. This condition is indicated by a flag in the SWPTABLE entry for the virtual page. The DMKPAGIO routine maintains a queue of CPEXBLOKs to be dispatched when the pending page I/O is complete. The CPEXBLOK for the page in transit is located and a new CPEXBLOK, representing the current request, is chained to it.

Before exiting to wait for the paging operation to complete, DMKPTRAN checks to see if the deferred return (DEFER option) has been specified. If it has not, DMKPTRAN returns to the caller. If the DEFER option has been requested, DMKPTRAN exits to the dispatcher to wait for page I/O completion. When the requested page has been read into real storage, the list of CPEXBLOKs are unstacked first-in-first-out to satisfy all requests for the page that arrived while it was in transit.

If a page is not in transit, a page frame of real storage must be allocated to fill the request. Before the allocation routine is called, a test is made to see if the caller wishes the return to his routine or to be delayed until after the requested page is available. If the DEFER option is not requested, DMKPTRAN returns to the caller after first building and stacking a CPEXBLOK that allows processing of the page request to be continued the next time the dispatcher (DMKDSPCH) is entered.

DMKPTRAN next calls the FREELIST manager (DMKPTRFR) to obtain the address of the next available CORTABLE entry. DMKPTRFR maintains a first-in-first-out list of the CORTABLE entries for those page frames that are immediately available for assignment. As DMKPTRFR releases these page frames, a check is made to see if the number of entries on the FREELIST has fallen below a dynamically maintained minimum value. If it has, the page selection routine (SELECT) is called to find a suitable page frame for placement in the FREELIST. The number maintained as the FREELIST threshold has a value equal to the number of users in queue1 plus the number of users in queue2 plus 1.

The FREELIST is replenished directly by users releasing virtual storage space. The page-out routine, DMKPGSPO, calls DMKPTRFT to place released page frames directly on the FREELIST. However, most replenishment is done via the page selection routine, SELECT. SELECT is called by DMKPTRFR when the FREELIST count falls below the current minimum, or when a user page is reclaimed from the FREELIST. In either case, the selection algorithm attempts to find a page to swap to auxiliary storage. The highest-priority candidates for a swap are those page frames whose CORTABLE entries appear on the FLUSHLST. SELECT attempts to take a flushed page frame before it takes a page frame from an active user. If such a page frame is found, it is checked to see if it has been changed since page-in. If it has not, it is placed in the FREELIST by DMKPTRFT; otherwise, it is scheduled for a swap-out by dequeuing the CORTABLE entry from the FLUSHLST, constructing a CPEXBLOK for dispatching after I/O completion, and exiting to DMKPAGIO by a GOTO. After the paging I/O is complete, the entry is placed on the FREELIST via a call to DMKPTRFT.

Once a page frame has been assigned, DMKPTRAN checks to see if a page-in is required. It usually is, and the DASD address of the virtual storage page must be obtained from the user's swap table entry and the I/O operation scheduled. However, if the page frame has not yet been referenced (as indicated by a DASD address of zero), the real main storage page frame is set to zero, and no page-in is required. After the page-in operation has been queued, DMKPTRAN exits to the paging I/O scheduler (DMKPAGIO), which initiates the paging operation and exits to the dispatcher (DMKDSPCH) to await the interruption.

Some requests for main storage page frames are handled differently from general virtual-to-real storage mapping. In particular, it may be necessary for CP to obtain additional free storage for control blocks, I/O lists, buffers, etc. This is handled by the free storage manager, which makes a direct call to DMKPTRFR to obtain the needed storage. Usually, this storage is immediately available (due to the page buffering technique previously described). However, if the FREELIST is exhausted, the request for free storage is recognized as a high-priority call and queued first on the list of those waiting for free page frames.

The real storage manager (DMKPTR) accumulates paging statistics that the scheduler (DMKSCH) uses to anticipate user storage requirements. A count of page-reads and page-writes is kept in each virtual machine's VMBLOK; the corresponding total counts for the system are kept in DMKPSA. A running total of the number of pages a virtual machine has resident, at each instance of page-read, is kept in the VMBLOK. A count of the number of times a virtual machine enters page-wait, because a page frame has been stolen from it, is also kept in the VMBLOK. The section entitled "Controlling of Multiprogramming" on page 182 under "Dispatching and Scheduling" on page 177 describes the use to which the scheduler puts these counts.

VM/SP Virtual=Real Option: The VM/SP virtual=real option involves the mapping in a one-for-one correspondence of a virtual machine storage area with an equivalent real storage area. For instance, virtual page 1 is in real page frame 1 and virtual page 20 is in real page frame 20. Virtual page 0 is relocated at the end of the virtual storage space because it cannot occupy real page frame 0.

The CP nucleus is altered at system generation to support the virtual=real option. Virtual machines with virtual=real (specially identified in the directory) can then logon and use the space reserved for this option. That space can be used by only one virtual machine at a time. Two virtual machines with the virtual=real capability cannot occupy the same space at the same time. The reliability and availability of an operating system running virtual=real on a 3081 processor can be enhanced when the TEST BLOCK instruction (TB) is used to validate the V=R storage area.

The virtual=real option allows the virtual machine to bypass the control program's CCW translation. This is possible because I/O from a virtual machine occupying a virtual=real space contains a list of CCWs whose data addresses reflect the real storage addresses. The restriction in this situation is that the virtual machine does not perform I/O into page frame 0 because this would perform a data transfer into real page frame 0. At the same time, it is assumed, and cannot be checked, that the virtual machine also does not attempt to do I/O beyond the bounds of its virtual addressing space. To do so would cause the destruction of either the CP nucleus, which resides beyond the virtual machine space, or another user's page.

Virtual 270X lines and sense operations from the virtual machine do not use the virtual=real function. These invoke CCW translation for the virtual enable/disable lines and the transfer of the sense bytes.

If the real I/O device is an MSS 3330V, then CCW translation is not bypassed since CP must still be able to recognize an MSS cylinder fault. See Appendix B, "VM/SP MSS Support" on page 409 for details.

The bypassing of CCW translation for the virtual machine occupying the virtual=real space is only invoked after the virtual machine has executed the SET NOTRANS ON command. This command can only be issued by the virtual machine occupying the virtual=real space. The command initiates the bypass of CCW translation. This option is automatically turned off if the virtual machine performs an explicit reset or an implied reset by performing a virtual IPL. During virtual machine IPL, I/O must be performed into page frame 0. For this reason, normal virtual IPL simulation assumes CCW translation in effect to accomplish the full simulation. Once the IPL sequence has completed, CCW translation can be bypassed by issuing the SET NOTRANS ON command.

When the virtual machine demands a page frame through normal use of CP's page tables, the paging routine recognizes the virtual=real capability. It then assigns the virtual page to the equivalent real page frame and does not perform a paging operation, because all these pages are resident and are never swapped out.

Note: The virtual machine running with virtual=real is still run in System/370 relocate mode.

Shadow table bypass, invoked by the SET STBYPASS command, allows VM/SP to eliminate the shadow tables for an operating system running in the V=R area. When CP runs a V=R user, the shadow table for the V=R user is identical to the virtual system's own page and segment tables, with the exception of page zero. CP relocates the virtual machine's page zero (via the shadow table) to the highest real address within the V=R area. When STBYPASS is turned on, CP modifies the virtual operating system's page table to relocate virtual page zero to the highest real address. It is then possible to dispatch the virtual machine with control register 1 pointing to the virtual page and segment tables.

The UNLOCK command has a VIRT=REAL operand that essentially releases the virtual=real area for normal system paging use. Once the area has been released, it can only be reclaimed for additional virtual=real operations only by an IPL of the VM/SP system. The size of the virtual=real area is an installation specification that is part of the special nucleus generation procedure that is outlined in the *VM/SP Installation Guide*. The size of the area must be large enough to contain the entire addressing space of whatever V=R machine wishes to occupy that space. A V=R machine can use a smaller space than is provided but cannot use a larger space without regenerating the CP nucleus.

DASD Storage Management

Any virtual machine's virtual storage pages that have been referenced but are not resident in real storage must be kept in slots on the DASD paging device. DASD page space is assigned only when the page is selected for a page-out. Certain DASD pages may also be marked read-only. Thus, the DASD address slot initially associated with the page should be considered to be the source of the page only. If the page is changed after it has been read into real storage, a new slot must be obtained when it is paged out. Examples of read-only pages are those which contain portions of pageable saved systems and pages which are part of a system spool file. Slots can be reassigned when DMKPTRAN finds that it must swap a page out to a movable-head DASD device. In this case, the old slot is released and the new slot is obtained.

Slot Allocation

If a new slot is required, DMKPGT is called to supply the address of an available slot. DMKPGT maintains a chain of cylinder allocation maps for each cylinder that has been assigned for either virtual storage or spool file paging. The allocation chains for spooling are kept separately from those used for paging so that they can be checkpointed in case of a system failure. However, in other respects they are the same. The allocation blocks for a given volume are chained from the ALOCBLOK or ALOFBLOK for the device on which the volume is mounted. The chains of cylinder and slot allocation blocks are initialized by DMKCPI. Each block on an allocation chain represents one cylinder of space assigned to paging, and contains a bit map indicating which slots have been allocated and which are available. Each block also has a pointer to the next allocation block on the chain, a cylinder number, and a record count. DMKPGT searches this list sequentially until an available slot is found; its DASD address is then determined and passed back to the calling routine. If DMKPGT cannot find a cylinder with a de-allocated slot, it enters the cylinder allocation phase. When an available cylinder is found, it constructs a page allocation block for this cylinder and allocates a page to the caller.

Cylinder Allocation

DMKPGT controls the paging and spooling I/O load of the system by allocating cylinders evenly across all available channels and devices. In order for a device to be considered available for the allocation of paging and spooling space:

- Its volume serial number must appear in the system's owned list.
- For count-key-data, it must have at least one cylinder of PAGE or TEMP allocated space marked as available in the cylinder allocation block which is located on cylinder 0, head 0, record 4.
- It must not be an MSS 3330V volume.
- For FB-512 DASD, it must have at least one page of PAGE or TEMP space marked as available in the allocation extent map located in blocks 3 and 4.

At system initialization time, CPINIT reads in the allocation records for each volume and constructs the chains of device allocation blocks from which DMKPGT allocates the cylinders. In managing the cylinder allocation, DMKPGT takes three factors into consideration: device type, device feature, device address, and possible status as a preferred paging device.

A request for a cylinder of virtual storage page space is satisfied by allocating space on a preferred paging cylinder or extent, provided that one exists on the system and that it has page space available. Preferred paging space is specified by the installation by volume allocation, and generally should be on devices on which excessive seek times do not occur. Typical preferred paging space would be on the IBM 2305 Fixed Head Storage facility as well as other DASD with fixed-head areas (such as, 3350). If the 2305 has preferred paging space allocated on it, it is possible to allocate some of its space for other high-priority data files without excessively degrading paging. An example of such usage would be for high activity read-only saved system pages that are not shared in real storage, and high-activity system residence disks.

It is also possible to designate space on movable-head DASD devices such as the 3310, 3330, 3340, 3350, 3370, 3375, 3380, and 2314/2319 Direct Access Storage facilities as preferred paging space. The module(s) so designated should not be required to seek outside of a relatively narrow cylinder band around the center of the paging areas. It is advisable to share the access arm of a movable-head device used for paging with only the lowest-usage data files.

If one or more volumes with preferred paging space are defined on the system, CP allocates all of the page space available on these before it allocates on any other available owned volumes. Within the class of volumes with preferred space, cylinders or extents are allocated according to the order specified at system generation time with the SYSFH and SYSMH parameters of the SYSORD Macro. Slots are first allocated from fixed-head areas; they are allocated in the order the installation specifies on the SYSFH parameter. If none is specified, the order is 2305, 3350, 3330, and 3340. If no fixed-head space is available, then slots will be allocated from moveable heads areas in the order specified on the SYSMH parameter of the SYSORD macro. If none is specified, the order is 3380, 3375, 3370, 3310, 3350, 3330, 3340, and 2314. Finally, if no preferred space is available, then slots will be allocated form non-preferred space in the order specified on the SYSTEMP parameter of the SYSORD macro. Cylinders for spooling space are not allocated from preferred space; they are allocated from nonpreferred space according to the order the installation defines with the SYSTEMP parameter. Allocation on a given device is done from the relative center of the volume outward, a cylinder at a time in a zigzag fashion in an attempt to minimize seek times.

When a request to allocate a slot for virtual storage paging is received by DMKPGTGT and the slot must be allocated on a moveable head (2314/2319, 3310, 3330, 3340, 3350, 3370, 3375, or 3380) device, a cylinder and slot are selected in the following manner:

1. CP tries to allocate a space on the cylinder at which the arm on the selected device is currently positioned.

2. If slots are not available on the current cylinder, CP tries to allocate space on a cylinder for which paging I/O has been queued.

3. If the above conditions cannot be met, CP allocates space as close to the center of the volume as is possible.

Before DMKIOSQR is called, the queue of IOBLOKs currently scheduled on the device is examined. If paging I/O has already been scheduled on a device, the paging channel programs are slot-sorted and chained together with TICs.

Paging I/O

DMKPAGIO handles all input/output requests for virtual storage and spooling pages. DMKPAGIO constructs the necessary task blocks and channel programs, expands the compressed slot addresses, and maintains a queue of CPEXBLOKs for pages to be moved. Once the I/O scheduled by DMKPAGIO completes, it unchains the CPEXBLOKs that have been queued and calls DMKSTKCP to stack them for execution. DMKPAGIO is entered by a GOTO from:

- DMKPTRAN to read and write virtual storage pages
- DMKRPA to read and write virtual storage spool buffers.

In either case, all that needs to be passed to DMKPAGIO is the address of the CORTABLE entry for the page that is to be moved, the address of a SWPTABLE entry for the slot, a read or write operation code, and the address of a CPEXBLOK that is to be stacked for dispatching after the I/O associated with the page has completed. DMKPAGIO obtains an IOBLOK and builds a channel program to do the necessary I/O, and uses the device code that is part of the page address to index into the system's OWNDLIST and locate the real device to which the I/O request should be directed. If the device is capable of rotational position sensing, the required sector is computed and a SET SECTOR command is inserted into the channel program. The real SIO supervisor DMKIOSQR is then called to schedule the operation on the proper device.

When the interruption for the paging operation is processed by the primary I/O interruption handler, the IOBLOK primary interruption handler, the IOBLOK that controls the operation is unstacked to the interruption return address, (DMKPAHIO). DMKPAHIO unchains the CPEXBLOKs that are queued to DMKPAGQ, and stacks the queued CPEXBLOKs (by calls to DMKSTKCP) in the order in which they were received. The address of the real page frame is filed into the appropriate page table entry, and the pointers denoting the ownership of the real page frame are filed into the CORTABLE entry by the processing routines in DMKPTRAN. If a fatal I/O error occurred for the related page frame, the CPEXBLOKs associated with it are flagged, and the dispatcher, DMKSDPCH, sets a nonzero condition code when it activates the pending task. The error recovery followed depends on the operation being performed. Paging I/O errors associated with spooling operations are discussed in "DASD Errors During Spooling" on page 202, while errors associated with virtual storage paging operations are discussed later in "Virtual Storage Paging Error Recovery" on page 151.

DMKPAHIO maintains its own subpools of preformatted paging IOBLOKs. As I/O operations complete, their IOBLOKs are added to a list of available blocks; as new blocks are needed, they are taken from this list. If the list is empty, DMKFREE is called to obtain storage for a new block.

Paging Subsystem

The paging subsystem has three major components that have resource optimization algorithms associated with them:

- The page replacement and page selection algorithm that manages the allocation of real storage frames and selects which virtual page to replace
- An algorithm for the allocation of DASD backing store pages
- An algorithm for ordering the queue of page I/O requests.

Page Replacement and Page Selection Algorithm

VM/SP is a demand paging system. Programs run in virtual storage and when a storage reference is made to a virtual page not currently in real storage, a page fault occurs. A page fault is a program interruption that occurs when a page marked `not in real storage` is referred to by an active page. This page fault represents a demand for a real storage frame in which to place the virtual page. The page replacement algorithm chooses which real storage frame will be allocated to fulfill such a demand. If all real frames in real storage are occupied by other virtual pages, a real frame can only be obtained by replacing one of those virtual pages. The selection of which virtual page to replace is carried out by the page selection algorithm.

The scheduler aids the page selection algorithm by notifying it of virtual machines that are no longer eligible for dispatching (either because they have been dispatched, or because they are being held suspended in the eligible list). The scheduler calls the page reset routine when a virtual machine is dropped from a queue and does not immediately reenter the dispatch list. Under heavy paging loads, it is the responsibility of the page reset routine to group all in-storage virtual pages belonging to the virtual machine; it groups them on an available (or flush) list for easy selection by the page replacement algorithm.

The page reset routine cycles through the virtual machine's segment table looking for valid segment entries. When it finds a valid entry, it turns on the segment table entry invalid flag and the page reset routine begins to process the page table associated with that segment table entry. The page table header is timestamped, and if it is a shared segment, the active segment table entry count is decreased. For a shared segment, if the active count is still greater than zero, no further processing is done. If the count has decreased to zero, for a shared segment, processing continues as if it were a private segment. Each page table entry in a segment is then examined for an in-storage page. If one is found, it has its reference bit reset to zero. In addition, if the heavy paging condition flag has been set, the page table entry is marked invalid, and the real page is placed on the flush list (FLUSHLST) in last-in-first-out order.

Note: The SET QDROP *userid* OFF command may be used to bypass page processing for selected virtual machines.

Page Selection

The page replacement/page selection algorithm must find a real frame to satisfy a demand for a virtual page. It first attempts to satisfy the demand with a page from the flush list. The flush list contains virtual pages (if any) that belong to virtual machines that are not eligible for dispatching, and therefore are not being used.

Note: A virtual machine may reenter the dispatch list after its pages have been placed on the flush list. If the virtual machine attempts to access any of those pages, they will be reclaimed. In order to increase the chances of a virtual machine reclaiming its pages, pages are selected from the FLUSHLST in last-in-first-out order.

If a page must be selected, it is better to use a page from a virtual machine that became ineligible for dispatching very recently. This recently ineligible virtual machine is less likely to become eligible for dispatching than a virtual machine that has been ineligible a long time. Therefore, the pages belonging to the recently ineligible virtual machine are less likely to be reclaimed than those belonging to the virtual machine that has been ineligible a long time.

For example, virtual machine A becomes ineligible for dispatching at 8:00. Virtual machine B becomes ineligible at 8:05. At 8:06 a page is required and is selected from virtual machine B's pages. At 8:08 virtual machine A becomes eligible again and all its pages are reclaimed. Virtual machine B does not become eligible again until 8:13.

If no pages are found on the FLUSHLST, the selection algorithm examines each virtual page in real storage, searching for an available page that does not have its reference bit on. It begins the search at the first available virtual page at the high end of real storage and searches by descending page address. When it reaches the lowest available page address, it starts again from the top of storage. When a page has been found, that page address minus one is checkpointed. The next time the selection algorithm is invoked, it will start from the checkpointed address. As the selection process proceeds, those pages that were not selected have their reference bits turned off. When the selection algorithm is operating in this mode, a virtual page must be referenced at least once per reset cycle (loop around real storage) to avoid selection.

Backing Store Allocation Algorithm

There are two parts to the algorithm for allocation of a DASD page record. The first is to find the optimal device on which to allocate a record. The second is then to optimize the record allocation on a particular device.

Device Selection

CP maintains the DASD device chain in two parts. The major part is the ordering of all devices by type and by the TEMP/PAGE classification. All PAGE devices are ordered before all TEMP devices. The device type fixed-head preferred paging default ordering is: 2305, 3350, 3330, 3340. The device type default for preferred moveable head paging is: 3380, 3375, 3370, 3310, 3350, 3330, 3340, and 2314. A system generation macro, SYSORD, allows the customer to reorder DASD to his own preference. All devices of the same type are chained together off the primary chain unless the customer specifies differently via the SYSORD macro. CP attempts to allocate a page record on the highest-level device until all devices at that level are full and then it tries the next lower device type. Within a particular device type, CP allocates records in a round-robin manner, attempting to evenly distribute the allocated records.

Cylinder Selection

Once a device is selected, CP must determine on which cylinder to allocate a record on that device. CP maintains a chain of cylinder record maps, one for each allocatable cylinder on the device. For 2305 devices, CP attempts to keep cylinder map blocks at the head of the chain. The only optimization done for a 2305 is an attempt to minimize the amount of processor time involved in the allocation process. For movable-arm DASD (that is, not 2305), CP attempts to allocate the first available record found according to criteria defined under "Cylinder Allocation" on page 143.

Page Selection Routine Support

Whenever a changed page is selected for replacement, it must first be copied onto DASD before the real page can be made available. In cases where there is already a DASD record allocated for the page and it is on a movable-arm DASD, the page selection routine deallocates the old record and requests that a new record be allocated. This occurs each time a page is to be written and its current backing-store location is on a movable-arm DASD. Although this represents overhead in terms of processor use, it is justified because it should minimize arm movement and reduce page wait time.

Page I/O Request Queueing Algorithm

The ordering of page I/O requests that are chained together for initiation with one SIO is done on a priority ordering basis. The priority is:

1. In-queue requests
2. Not-in-queue requests
3. Reads
4. Writes
5. Q1 requests
6. Q2 requests.

PCI flags are set for page I/O requests. For non-2305 requests, there is an interruption after each request. For 2305 requests, the PCI flag is set so that there is one interruption for each revolution of the drum (one interruption for every three requests).

Note: For installations that are much more constrained by a page I/O bottleneck (as opposed to processor bottleneck), the 2305 PCI mode can be changed to operate in the same way as the non-2305 processing, that is, by allowing an interruption immediately after each request. The SET SRM PCI DISK command causes the PCI flag to be set so there is one interruption for each 2305 page request. SET SRM PCI DRUM changes it back to the default mode of operation.

Page Migration and Swap Table Migration

Both page and swap table migration are invoked through the performance monitoring routine (DMKSTP). The performance monitoring routine is invoked at regular intervals (between 10 and 60 seconds). It has the responsibility of determining whether conditions warrant invoking page migration and swap table migration. Since swap table migration only migrates tables for segments with no unmigrated pages, page migration is always invoked before swap table migration.

Page Migration

Page migration is invoked via a CP request block stacked by DMKSTP. DMKSTP invokes page migration if:

- All fixed-head preferred paging areas are full, or
- The specified percentage of moveable head preferred paging area is full, or
- Swap table migration is to be invoked, and
- If and only if at least 10 minutes have elapsed since the last time DMKSTP invoked page migration.

The objective of the page migration routine (DMKPGM) is to move inactive pages out of the preferred fixed-head paging areas. To do this, it relies on the PAGTABLE header timestamp (PAGSTMP). Page migration scans each virtual machine's segment table for entries that have both valid page table pointers and the invalid flag set (SEGINV). When such an entry is found, the timestamp for the corresponding page table is checked to see if at least 12 minutes have elapsed since the last time it was active. If the segment entry/page table meet these criteria, then all pages in that segment currently in a preferred fixed-head paging areas migrate to a nonpreferred area.

After scanning all segment table entries of all virtual machines, page migration either exits, continues on to swap table migration, or repeats the whole scan with a shorter time limit. If page migration was invoked by a CP command, it stacks a CP request block for swap table migration after the first scan and exits. Otherwise, if less than 12% of the preferred area has been made available, the page migration halves the time limit (initially 12 minutes) and repeats the scan. If necessary, the scan is made a total of three times (12 minutes, 6 minutes, and 3 minutes). If, after three scans, the availability criterion still has not been met, and there are preferred fixed-head devices, page migration switches to DRUM fair share allocation mode. Before exiting, page migration checks to see if DMKSTP has requested that swap table migration be invoked via a CP request block.

Page migration by command for a specific virtual machine is similar. In this case, only the specified virtual machine's segment table is scanned. Also, for a specific virtual machine, no time limit test is made. As long as the segment table entry criterion is satisfied, pages in that segment are migrated.

As page migration proceeds through the virtual storage tables, all pages found that are allocated on movable-arm DASD have their CP backup change flag turned on. This causes the page to be deallocated from its current position and a new allocation made the next time the page is brought into storage. This should migrate the page up to the best available position. At worst, it might be re-allocated in the same location.

Swap Table Migration

Swap table migration is always invoked via a CP request block stacked by page migration. Page migration invokes swap table migration if: 1) it was invoked by the MIGRATE command or 2) it was invoked by DMKSTP and DMKSTP also specified swap table migration.

DMKSTP requests swap table migration if the following conditions are met:

- There are extended free storage pages.

- The space occupied by page tables and swap tables is equivalent to at least 12% of nonextended free storage.

- There is a high paging load on the system.

Swap table migration scans the segment table in a manner similar to page migration. It performs one scan with a time limit of three minutes (except for migrating a specific virtual machine by command, where no time limit is used). Segments are selected that:

- Meet the time criterion
- Do not have any unmigrated or in-storage pages
- Do not have a named system pointer.

For selected segments, the swap table for the segment is copied to a 4K page buffer, which is then written to disk. The space occupied by both the page table and swap table is returned to available free storage. The associated segment table entry has its page table pointer set to 0 and the SEGMIG flag turned on.

The swap table is brought back into storage whenever a reference is made to that segment. The page translation routine (DMKPTR) always calls the swap table migration routine (DMKSTR) whenever a reference is made to a segment table entry with a zero page table pointer. If appropriate, DMKSTR calls DMKBLD to have a page table and a swap table built for that segment. If the swap table has been migrated, the 4K buffer page is brought into storage and the swap table information is copied into the swap table.

Virtual Storage Paging Error Recovery

Errors encountered during virtual storage (as opposed to spooling) paging operations can generally be classified as either soft or hard errors. Soft errors allow the system to continue operation without delay or degradation. Hard errors can cause noticeable effects such as the abnormal termination of user tasks (abend) and response degradation. Errors that are successfully retried or corrected are known only to the I/O supervisor and the I/O error retry and recording routines; they appear to the second level interruption handlers (such as WAITPAGE) as if the original operation completed normally.

Soft Error Recovery: An I/O error that occurs on a page swap-out is considered to be a soft error. DMKPTRAN calls DMKPGTPG to assign a different DASD page slot and the page is re-queued for output. The slot that caused the error is not de-allocated, and thus is not assigned to another virtual machine. All other uncorrectable paging errors are hard because they more drastically affect system performance.

Hard Error Recovery: Hard paging errors occur on either I/O errors for page reads or upon exhausting the system's spooling and paging space. Recovery attempted on hard errors depends upon the nature of the task for which the read was being done. If the operation was an attempt to place a page of a virtual machine's virtual storage into real storage, the operation of that particular virtual machine is terminated by setting the page frame in error to zero and placing the virtual machine in console function mode. The user and operator are informed of the condition, and the page frame causing the error is not de-allocated, thereby ensuring that it is not allocated to another user.

The control program functions that call DMKPTRAN (such as spooling, pageable control program calls, and system directory management) have the option of requesting that unrecoverable errors be returned to the caller. In this case, the CP task may attempt some recovery to keep the entire system from terminating (abend). In general, every attempt is made to at least allow the operator to bring the system to orderly shutdown if continued operation is impossible.

Proper installation planning should make the occurrence of a space exhaustion error an exception. An unusually heavy user load and a backed-up spooling file could cause this to happen. The operator is warned when 90% of the temporary (paging/spooling) space in the system is exhausted. He should take immediate steps to alleviate the shortage. Possible remedies that exist include preventing more users from logging on and requesting users to stop output spooling operations. More drastic measures might include the purging of low-priority spool files. If the system's paging space is completely exhausted, the operation of virtual machines progressively slows as more and more users have paging requests that cannot be satisfied and operator intervention is required.

Virtual Relocation

CP provides the virtual machine the capability of using the DAT feature of the real System/370. Programming simulation and hardware features are combined to allow usage of all of the available features in the real hardware, (that is, 2K or 4K pages, 64K or 1M segments).

For clarification, some term definitions follow:

First-level Storage: The physical storage of the real CPU, in which CP resides.

Second-level Storage: The virtual storage available to any virtual machine, maintained by CP.

Third-level Storage: The virtual storage space defined by the system operating in second-level storage, under control of page and segment tables which reside in second-level storage.

Page and Segment Tables: Logical mapping between first-level and second-level storage.

Virtual Page and Segment Tables: Logical mapping between second-level and third-level storage.

Shadow Page and Segment Tables: Logical mapping between first-level storage and third-level storage.

A standard, nonrelocating virtual machine in CP is provided with a single control register, control register zero that can be used for:

- Extended masking of external interruptions
- Special interruption traps for SSM
- Enabling of virtual block multiplexing.

A virtual machine that is allowed to use the extended control feature of System/370 is provided with a full complement of 16 control registers, allowing virtual monitor calls, PER, extended channel masking, and dynamic address translation.

An extension to the normal virtual-machine VMBLOK is built at the time that an extended control virtual machine logs onto CP. This ECBLOK contains the 16 virtual control registers, 2 shadow control registers, and several words of information for maintenance of the shadow tables, virtual CPU timer, virtual TOD clock comparator, and virtual PER event data. The majority of the processing for virtual address translation is performed by the module DMKVAT, with additional routines in DMKPRG, DMKPRV, DMKDSP, DMKCDB, DMKLOG, DMKUSO, and DMKPTR. The simulation of the relocation-control instructions (that is, LCTL, STCTL, PTLB, RRB, and LRA) is performed by DMKPRV or DMKFPS. These instructions, with the exception of LCTL and STCTL, are not available to virtual machines which are not allowed the extended control mode.

When an extended-control virtual machine is first active, it has only the real page and segment tables provided for it by CP and operates entirely in second-level storage. DMKPRV examines each PSW loaded via LPSW to determine when the virtual machine enters or leaves extended control or translate mode, setting the appropriate flag bits in the VMBLOK. Flag bits are also set whenever the virtual machine modifies control registers 0 or 1, the registers that control the dynamic address translation feature. DMKDSP also examines PSWs that are loaded as the result of interruptions to determine any changes in the virtual machine's operating mode. The virtual machine can load or store any of the control registers, enter or leave extended control mode, take interruptions, etc., without invoking the address translation feature.

If the virtual machine, already in extended control mode, turns on the translate bit in the EC mode PSW, then the DMKVATMD routine is called to examine the virtual control registers and build the required shadow tables. (Shadow tables are required because the real DAT hardware is capable of only a first-level storage mapping.) DMKVATMD examines virtual control registers 0 and 1 to determine if they contain valid information for use in constructing the shadow tables. Control register zero specifies the size of the page and segment the virtual machine is using in the virtual page and segment tables. The shadow tables constructed by DMKVATMD are always in the same format as the virtual tables.

The shadow segment table is constructed in first-level storage and initialized to indicate that all segments are unavailable. Flags are maintained in the VMBLOK to indicate that the shadow tables exist. DMKVATMD also constructs the shadow control registers 0 and 1. Shadow control register 0 contains the external interruption mask bits used by CP, mixed with the hardware controls and enabling bits from virtual control register 0. Shadow control register 1 contains the segment table origin address of the shadow segment table.

When the virtual machine is operating in virtual translate mode, CP loads the shadow control registers into the real control registers and dispatches the user. The immediate result of attempting to execute an instruction is a segment exception, intercepted by DMKPRG and passed to DMKVATSX. DMKVATSX examines the virtual segment table in second-level storage. If the virtual segment is not available, the segment exception interruption is reflected to the virtual machine. If the virtual segment is marked available, then DMKVATSX:

- Allocates one full segment of shadow page table, in the format specified by virtual control register 0
- Sets all of the page table entries to indicate page not in storage
- Marks the segment available in the shadow segment table
- Redispatches the virtual machine via DMKDSP.

Once again, the immediate result is an interruption, which is a paging exception and control is passed to DMKVATPX. DMKVATPX references the virtual page table in second-level storage to determine if the virtual page is available. If the virtual page is not available, the paging interruption is reflected to the virtual machine. However, if the virtual page is marked in storage, the virtual page table entry determines which page of second-level storage is being referenced by the third-level storage address provided. DMKVATPX next determines if that page of second-level storage is resident in first-level storage at that time. If so, the appropriate entry in the shadow page table is filled in and marked in storage. If not, the required page is brought into first-level storage via DMKPTRAN and the shadow page table filled in as above.

As the virtual machine continues execution, more shadow tables are filled in or allocated as the third-level storage locations are referenced. Whenever a new segment is referenced, another segment of shadow page tables is allocated. Whenever a new page is referenced, the appropriate shadow page table entry is validated, etc. No changes are made in the shadow tables if the virtual machine leaves translate mode (usually via an interruption), unless it also leaves extended control mode. Dropping out of EC mode is the signal for CP to release all of the shadow page and segment tables and the copy of the virtual segment table.

There are some situations that require invalidating the shadow tables constructed by CP or even releasing and reallocating them. Whenever DMKPTR swaps out a page that belongs to a virtual relocating machine, DMKVATSI is called to selectively invalidate the shadow page tables. If the stolen page is below the high-water mark, the shadow page table entry for the stolen page is invalidated. If the stolen page is above the high-water mark and virtual machine assist is on, a bit is set in the VMBLOK to indicate that all of the shadow page tables above the high-water mark must be invalidated. The actual invalidation is handled by DMKVATAB, called from DMKDSP when the virtual machine is about to be dispatched. If the stolen page is above the high-water mark and virtual machine assist is off, the shadow page tables are scanned to selectively invalidate shadow page table entries that map to the real page being stolen.

The other situations which cause shadow table invalidation arise from the simulation of privileged instructions in DMKPRV or DMKFPS. Flags are set in the VMBLOK whenever the virtual machine loads either control register 0 or 1, and DMKPRV calls DMKVATAB to perform whatever maintenance is required. When control register 1 is loaded by the virtual machine, DMKVATAB scans the chain of STOBLOKs to see if shadow tables are already allocated for the value in virtual control register 1. If a matching STOBLOK is found, it is requeued as the first in the STOBLOK chain and the virtual machine can be redispatched. If a matching STOBLOK is not found, and the number of STOBLOKs is equal to the maximum STOBLOK count, the last STOBLOK in the chain is reused by invalidating the entire shadow table and then queuing it first on the STOBLOK chain. If the number of STOBLOKs is less than the maximum STOBLOK count, a new STOBLOK is acquired and initialized, and placed first in the STOBLOK chain. When control register 0 is loaded, DMKVATAB examines the relocation-architecture control bits to determine if they have changed, (such that the format of the virtual page and segment tables no longer matches that of the shadow tables). If the format has not changed, the shadow tables are left intact; otherwise, all of the shadow tables must be returned to free storage and another set, in the new format, must be allocated and initialized. The same actions can result from modifying the control registers via the CP console functions, in which case DMKVATAB is called from DMKCDB. The privileged operation, PTLB, causes the shadow page tables to be invalidated above the high-water mark because the shadow tables are the logical equivalent of the translation look-aside buffer.

The privileged instruction LRA is simulated via DMKVATLA, which searches the virtual page and segment tables to translate a third-level storage address to a second-level storage address, returning a condition code indicator to DMKPRV, or forcing an interruption if the tables are incorrectly formatted.

Most error situations that occur in the virtual machine are handled by means of the extended program interruptions associated with the real address translation hardware. Whenever a virtual relocating machine loads control registers 0 or 1 with an invalid value, DMKVAT releases all of the shadow tables exactly as if the hardware controls had changed. The shadow control registers are set valid, with the shadow segment table re-allocated at a minimum size and all segments marked unavailable. Flag bits are set in the VMBLOK to indicate that the shadow tables are artificially valid, and DMKVATSX reflects a translation specification exception to the virtual machine as soon as it is dispatched. While it is possible for the virtual machine to enter an interruption loop (if the new PSW is also a translate mode PSW), the cited process prevents the occurrence of a disabled loop within CP, which would result if the virtual machine is never dispatched.

Extended Storage Key Support

DMKPRV checks the status of the reference and change bits in the virtual storage keys, which involve the privileged instructions ISK, RRB, and SSK.

For SSK, CP increases simulation counter DMKPRVEK, sets general purpose register 0 to X'0C' and branches to subroutine CKCR0B7 to simulate the instruction.

For ISK, CP increases the simulation counter DMKPRVIK and sets general purpose register 0 to X'04'. DMKPRV uses BAL linkage to invoke subroutine CKCR0B7.

For RRB, CP increases the simulation counter DMKPRVRR and sets general purpose register 0 to X'08'. DMKPRV uses BAL linkage to invoke subroutine CKCR0B7.

Subroutine CKCR0B7 examines bit 7 of CPCREG0 in the PSA to determine the type of real storage frames that are installed on the processor. If bit 7 of CPREG0 is B'1', this indicates that the storage simulated for this virtual machine consists of frames that are protected by single (4K) storage protection keys. CP does the following:

- Examines bit 7 of virtual control register 0 (VMVCR0 if in BC mode, or EXTCR0 if in EC mode).
- If virtual control register 0 bit 7 is B'1', DMKPRV simulates the key operations for the virtual machine.
- If virtual control register 0 bit 7 is B'0', CP returns a special operation exception to the virtual machine.

If CPCREG0 is B'0', this indicates that the storage for this virtual machine consists entirely of frames that are protected by two (2K) storage protection keys. DMKPRV simulates the key operation.

Free Storage Management

DMKFRE and DMKFRT are responsible for the management of free storage, and CP uses them to obtain and release free storage for I/O tasks, CCW strings, various I/O buffers, etc. They are used for practically all such applications except real channel, control unit, and device blocks, and the CORTABLE.

Blocks sizes of 33 doublewords or less, constituting about 99 percent of all calls for free storage, are grouped into 11 subpool sizes (ranging from 3 to 33 doublewords in steps of 3), and are handled by LIFO (push-down stack) logic. Blocks of greater than 33 doublewords are strung off a chained list in the classic manner.

When subpools are exhausted, small blocks are generally obtained from the first larger block at the end of available free storage. Large blocks, on the other hand, are obtained from the high-numbered end of the last larger block. This procedure tends to keep the volatile small subpool blocks separated from the large blocks, some of which stay in storage for much longer periods of time; thus, undue fragmenting of available storage is avoided.

DMKFRE initially starts without any subpool blocks. They are obtained from DMKFREE and returned to DMKFRET on a demand basis.

The various cases of calls to DMKFREE for obtaining free storage, or to DMKFRET for returning it, for subpool sizes and large sizes, are handled as follows:

Calling DMKFREE for a Subpool

Subpool Available: If a call for a subpool is made and a block of the suitable size is available, the block found is detached from the chain, the chain patched to the next subpool block of the same size (if any), and the given block returned to the caller.

Subpool Not Available: If a block of suitable size is not available when a call to DMKFREE is made for a subpool, the chained list of free storage is searched for a block of equal or larger size. The first block of larger or equal storage is used to satisfy the call (an equal-size block taking priority), except that blocks within the dynamic paging area are avoided if at all possible. If a block that is equal or larger cannot be found on the free storage chain, a check is made (starting with the largest subpool size, 33 doublewords, and working downwards) to see if a larger subpool block can be split into the size requested and another subpool size. If one is found, the larger block is detached from its subpool and split. The requested block is returned to the caller, and the remaining block is attached to the appropriate subpool. If this effort fails, then DMKPTRFR is called to obtain another page frame of storage from the dynamic paging area (which is merged into the large storage chain), and the process is repeated to obtain the needed block.

Calling DMKFREE for a Large Block

If a call to DMKFREE is made for a block larger than 33 doublewords, the chained list of free storage is searched for a block of equal or larger size. Blocks within the dynamic paging area are avoided if at all possible. If an equal-size block is found, it is detached from the chain and given to the caller. If at least one larger block is found, the desired block size is split off the high-numbered end of the last larger block found, and given to the caller. If no equal or larger block is found, DMKPTRFR is called to obtain another page frame of storage from the dynamic paging area (which is merged into the free storage chain), and the above process is repeated (as necessary) to obtain the needed block.

Calling DMKFRET for a Subpool

DMKFRET processes the CP Assist Fret instruction (E601) to return free storage. If the microcode cannot return the block to an appropriate subpool, control is passed to DMKFRTT to handle the request.

If a subpool block is given back via a call to DMKFRET, the block is attached to the appropriate subpool chain on a LIFO (push-down stack) basis, and return is made to the caller. If the block was in a page within the dynamic paging area, it is also placed in a subpool chain. (However, subpool storage is returned, via a call to DMKFRTRS, to the regular free storage chain once every hour or when a user logs off. DMKFRTRS then calls DMKFRTSN to search the free storage chain for page frame sized blocks needing to be returned to the dynamic paging area.)

Calling DMKFRET for a Large Block

If a block larger than 33 doublewords is returned via DMKFRET, control is passed to DMKFRTT where the block is merged appropriately into the regular free storage chain. Then a check is made to see if the area given back (after all merging has been done) is a page frame within the dynamic paging area. If so, DMKPTRFT is called to return the block to the dynamic paging area for subsequent use. If the block is returned by specific modules known to use large blocks frequently for very short periods of time, DMKPTRFT is purposely not called, to avoid continuous extending and unextending of free storage over very short time intervals.

Free Storage Page Frame Allocation

The number of page frames allocated to free storage depends upon:

- The real machine storage size
- The RMSIZE operand specified in the SYSCOR macro at system generation time
- The FREE operand in the SYSCOR macro
- The number of unusable or inaccessible storage frames detected during CP initialization (3081 processors only).

The storage size used by VM/SP is the smaller of the real machine storage size and the RMSIZE value.

If the FREE operand was not included in the SYSCOR macro statement for DMKSYS, the default number of fixed free storage pages allocated at IPL time for the first 256K of storage is 3 and 1 page for each 64K thereafter, not including V=R size, if any.

If the FREE operand was included in the SYSCOR macro statement for DMKSYS, that value is the number of fixed free storage page frames allocated at IPL time. If those pages represent an amount of free storage greater than 25% of the storage size (not including V=R size, if any) the default allocation is used.

CP FRET Trap

The CP FRET Trap detects the release of areas of free storage that were not assigned, previously released, or outside the boundaries of the storage given. Based on the value of the option &FRETRAP, the trap code is conditionally assembled in modules DMKCPI, DMKFRE, and DMKFRT. &FRETRAP can be found in OPTIONS COPY and has a default value of 0 for normal operations without the trap. The trap may be installed at system generation time. Refer to the VM/SP *Installation Guide* for the installation instructions.

When the trap is installed, DMKCPI disables CP Assists FREE, FRET, DSP1, DSP2, and UNTFR during system initialization. DMKFRE adds three doublewords to each free storage request, creating a trap extension area. The extension area contains:

- The status of the request. The status consists of the tag ALLO when the storage is allocated by DMKFRE or the tag FRET when the storage is released by DMKFRT.
- The saved size, in doublewords, of the requested free storage area.
- The starting address of the assigned free storage block.
- The return address of the module requesting the storage.
- The last three bytes of the calling module's name, if it is pageable.
- The user's VMBLOK address.
- An area for information about the module releasing the storage, initially cleared to zeroes.

For the format of the extension area, refer to the FREEXT control block in the VM/SP *Data Areas and Control Block Logic Volume 1 (CP).*

DMKFRT checks each request to release free storage for an ALLO tag. It checks at the address calculated by adding the size in bytes of the storage block being released to the address of the block. If the ALLO tag is found, the size of the free storage block being released is checked against the saved size in the extension area. If the sizes are equal, the ALLO tag is changed to FRET and the extension area is updated with information about the module releasing the storage. The requested free storage block, including the extension area, is then released.

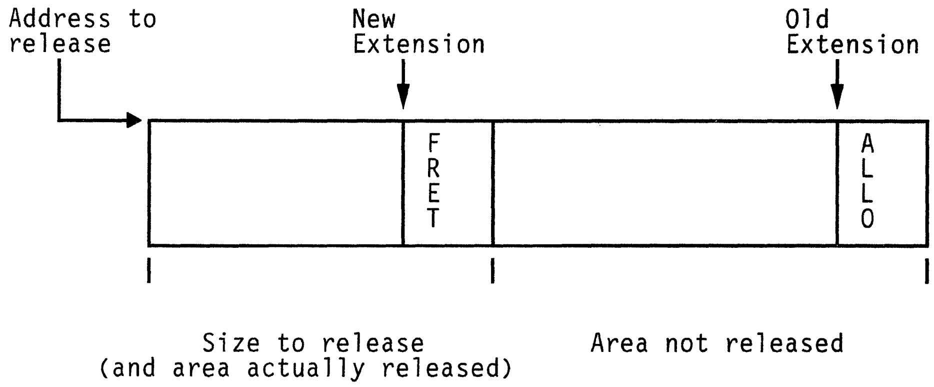
Trap Error Detection

When storage is released, for systems generated in uniprocessor mode (or in AP/MP mode and the second processor is not operating), the trap code in DMKFRT detects three types of errors. If the extension area cannot be located, the system abends with code FRT013. If the tag in the extension area is FRET instead of ALLO, indicating that the storage has already been released, the system abends with code FRT016. If the size of the free storage block being released does not match the saved size in the extension area, the system abends with code FRT015.

AP/MP Differences

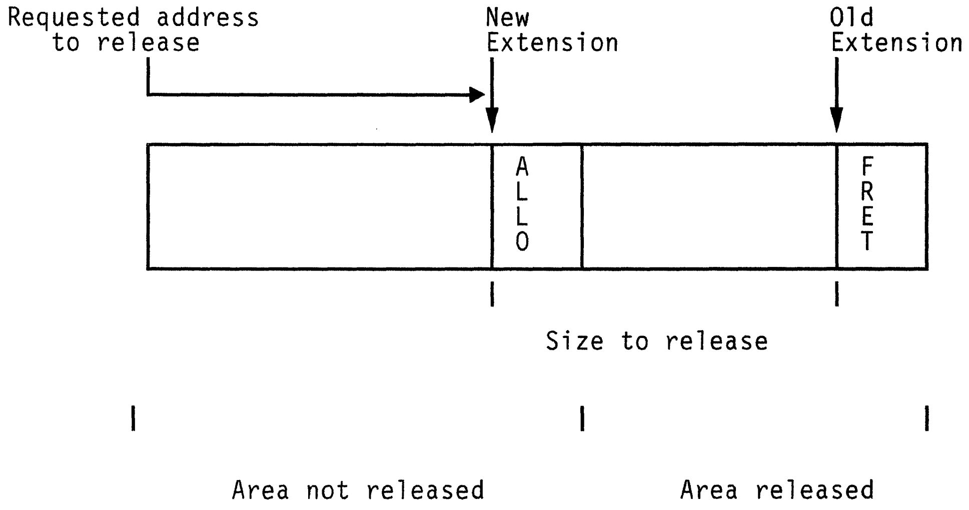
For systems generated in AP/MP mode, and the second processor is operating, the area released can be a split of the storage area given. Either the front or the rear portion of the free storage area may be released. Storage splits are allowed due to the way CP handles page and swap tables for shared systems in the AP/MP environment. With one call to DMKFREE, two sets of contiguous page and swap tables are obtained (one for each processor). Under some conditions, the page and swap tables for only one processor are released. The trap code allows only "valid" splits. A split is considered "valid" if both parts of the split free storage area are each the size of a contiguous page and swap table, and the saved size in the extension area is the size of two page and swap tables.

Valid Split: Release of Front Portion



When the front portion of the free storage block is being released, the extension area is updated: the saved size is adjusted to the size of a page and swap table, and the new address of the area not being released is stored. The information in the extension area is copied to create a new extension area. The new extension area is constructed within the front portion being released, using the last three doublewords. The new extension area is updated with the address of the front portion, information about the module releasing the storage, and the FRET tag. The front portion of the storage area is returned to the DMKFRELS chain.

Valid Split: Release of Rear Portion



When the rear portion of the free storage block is being released, the extension area is copied to create a new extension. The new extension area is constructed at the end of the front portion that was not requested to be released. The saved size, in both the old and the new extension areas, is adjusted to the size of a page and swap table. The old extension is updated with the new address of the rear portion actually released, information about the module releasing the storage, and the FRET tag. The new address of the rear portion is calculated by adding the requested address to release to the size in bytes of the new extension area. At this new address, the rear portion of the free storage block is returned to the DMKFRELS chain (including the old extension).

Trap Error Detection in AP/MP Mode

For systems generated in AP/MP mode when the second processor is operating, the trap code takes into account the possibility of storage splits while checking for errors. The trap code checks for the ALLO tag at the address calculated by adding the address of the free storage block to the size in bytes of the returned free storage area. If at this address a FRET tag is found, the system abends with code FRT016. If neither the ALLO or FRET tag is found, the trap code checks for an ALLO tag at that address plus the size in bytes of a page and swap table. The system abends with code FRT013 if the ALLO tag is still not found. If the ALLO tag is found at one of the two addresses checked, but the size of the free storage area to be released does not match the saved size in the extension area, then the trap checks for a "valid" split. Abend FRT015 occurs if the split is not "valid".

Initialization Overview

System initialization starts when the operator selects the DASD address of the CP system residence volume (SYSRES) and presses the IPL button. The System/370 hardware reads 24 bytes from the SYSRES (record 1 of cylinder 0) into location 0 of main storage. This record consists of an initial PSW and a channel program. The channel program reads the module DMKCKP from the SYSRES (record 2 of cylinder 0) into location x'1000' of main storage. (The load point for a V=R system is the V=R size plus x'2000', or DMKSLC plus x'1000'.) The hardware then loads the initial PSW from location 0 of main storage. This PSW causes execution to begin at the entry point DMKCKPT.

DMKCKP will load DMKSAV from the SYSRES and then pass control to DMKSAVRS. DMKSAVRS loads the CP nucleus from the nucleus area of the SYSRES into main storage starting at page 0 and ending with the pageable module immediately preceding DMKSAV in the CP load list. Next DMKSAV passes control to DMKCPINT.

| DMKCPINT performs the main initialization function. This includes:

- | Calling DMKSTANT to initialize main storage
- | Calling DMKMNTIO to mount the I/O devices which were defined in | DMKRIO and are available
- | Calling DMKSEGCP to set up CP's segment page, core, and swap tables
- | Building a LANGBLOK for the installation default language

- Calling DMKOPERC to locate the operator's console and to initialize the operator's VMBLOK

- Calling DMKTODIN to initialize the time-of-day clock

- Calling DMKCPJNT to continue system initialization

- Calling DMKUDROV to read any command class override records stored on the primary override cylinders.

DMKCPJNT calls DMKWRMST to perform a WARM, CKPT, FORCE or COLD start, DMKIDUMP to locate DASD space for system dumps, and DMKOPELO to LOGON the system operator. When control is returned from DMKCPJ, DMKCPI goes to the dispatcher, DMKDSPCH, to begin dispatching virtual machines.

See Figure 3 on page 24 for a more detailed description of CP initialization.

Warm Start Overview

Following an orderly system shutdown where warm start data has been saved by the checkpoint programs (DMKCKP, DMKCKD, DMKCKF, DMKCKH, DMKCKM, and DMKCKN) it is possible to start the system with a warm start. DMKWRMST is called by DMKCPJ during system initialization to perform a warm start. DMKWRMST reconstructs the system log message, the saved accounting ACNTBLOKs, the saved printer, punch, reader and delete SPOOL file blocks (SFBLOKs), the saved ALOCBLOK and RECBLOK data, and the saved SPOOL hold queue blocks (SHQBLOKs). In addition, if this is an automatic system re-IPL following a dump to DASD, DMKWRMST re-enables the terminals which were enabled when the system abended. DMKWRM then calls DMKCKSIN to initialize the checkpoint area and DMKWRNWM to checkpoint the SHQBLOKs and SFBLOKs reconstructed during the warm start. Finally DMKWRMST invalidates the first record of the warm start area so that another warm start cannot occur until the system undergoes an orderly shutdown.

Checkpoint Start Overview

If the system is unable to perform a warm start because of I/O errors or because no warm start data was saved from the last session, then a checkpoint (CKPT) start can be attempted. This option attempts to initialize the system using the information that has been dynamically checkpointed during system operation and stored on the checkpoint area. DMKWRMST is called by DMKCPJ for a CKPT start. DMKWRM calls DMKCKVWM to handle the CKPT start. DMKCKVWM restores the checkpointed real device information for system printers, punches, and readers, and reconstructs the SPOOL hold queue blocks (SHQBLOKs) and SPOOL file blocks (SFBLOKs) which were dynamically checkpointed during system operation. Finally, DMKWRMST invalidates the first record of the warm start area so that a warm start cannot occur until the system undergoes an orderly shutdown.

The system log message is not reconstructed as for a WARM start. Also, since ALOCBLOK and RECBLOK information is not saved on the checkpoint area, the pages used by the reconstructed SPOOL files must be allocated by following the SPOOL file links (SPLINKs) for each reconstructed SPOOL file. Because of this, a checkpoint start takes longer than a warm start.

Force Start Overview

If the system is unable to perform a checkpoint start because of I/O errors or invalid data on the checkpoint area, then a FORCE start can be attempted. A FORCE start works in the same way as a CKPT start, except DMKCKVWM skips any checkpoint record which contains unreadable or invalid data. For a CKPT start, DMKCKVWM loads a disabled wait state PSW (code 00E) if an invalid checkpoint record is encountered.

Cold Start Overview

A COLD start is usually performed only on the initial loading operation of a new version of the VM/SP system or when a hardware error prevented valid system checkpointing and shutdown. DMKWRMST is called by DMKCPJ during system initialization to perform a COLD start. No data is recovered from the checkpoint or warm start area. DMKWRMST calls DMKCKSIN to initialize the checkpoint area. Next DMKWRMST invalidates the first record of the warm start area so that a warm start cannot occur until the system undergoes an orderly shutdown.

Shutdown Start Overview

A SHUTDOWN start is performed in order to halt the initialization process. For example, if vital CP owned volumes are not mounted, then it might be desirable to mount those devices and then re-IPL without first doing a WARM, CHECKPOINT, FORCE or COLD start. For a SHUTDOWN start no processing of system warm start or checkpoint data is done. A disabled wait state PSW (code 006) is loaded.

System Shutdown and Auto Start Overview

When the operator or other authorized user issues the SHUTDOWN command without the REIPL option, DMKCPSSH moves “CPCP” into the CPID field of the PSA. “CPCP” tells the checkpoint program that a system shutdown is to be performed. If the SHUTDOWN command is issued with the REIPL raddr option, the alternate device address is stored in DMKDMPSD. DMKDMP uses the address to determine what device is the target of the IPL CCW.

When the operator or other authorized user issues the SHUTDOWN command with the REIPL option, DMKCPSSH moves “WARM” into the CPID field of the PSA. “WARM” tells the checkpoint program that a system shutdown is to be performed and then an automatic warm start is to be performed.

DMKCPSSH then goes to DMKDMPRS. DMKDMPRS issues an IPL CCW to read the IPL sequence and, subsequently, DMKCKP from the SYSRES just as during system initialization. DMKDMPRS then loads the initial PSW from location 0 of main storage and DMKCKPT gets control.

DMKCKPT reads in the rest of the checkpoint programs (DMKCKD, DMKCKF, DMKCKH, DMKCKM, and DMKCKN). DMKCKPT calls DMKCKDEV to drain pending I/O interrupts, and issue a HIO to all available devices. DMKCKPT calls DMKCKFIL to save the addresses of enabled terminals, the status of the system operator, device and user accounting cards, the system log message, printer, punch, reader, and delete SPOOL file blocks (SFBLOKs), open CPTRAP and monitor SPOOL files, the allocation RECBLOKs, and the SPOOL hold queue blocks (SHQBLOKs). DMKCKFIL saves this information on the warm start area for recovery by DMKWRM during a warm start. DMKCKP calls DMKCKMSV to save virtual machines which were enabled for VMSAVE. DMKCKMSV saves the virtual machines on the CP owned DASD as specified in the system name table, DMKSNT. If CPID = "CPCP" (i.e. SHUTDOWN was requested), then DMKCKPT loads a disabled wait state PSW (code 008) and the system shutdown is complete. If CPID = "WARM" (i.e. SHUTDOWN REIPL was requested), then DMKCKPT will load DMKSAV, set CPID to "WRM" and transfer control to DMKSAVRS in the same way that a normal system initialization is performed. DMKCPINT will get control from DMKSAV and perform the same functions as for normal system initialization, but since CPID is "WRM," the operator will not be requested to change the time of day clock, nor choose the type of start to perform. In fact, it is not necessary for an operator to be present during this automatic re-IPL. The system will automatically perform a WARM start (see Figure 4 on page 25).

When the operator or other authorized user issues the SHUTDOWN command with the POWEROFF option, DMKCPSSH turns on a bit to indicate that power off and automatic IPL are requested. An automatic IPL bit is turned on in the first warm start record in DMKCKH. DMKCKP then issues a power off diagnose instruction. If an operating system is running as a virtual machine, no power off diagnose instruction is issued and a normal shutdown takes place.

When a processor is powered on and VM/SP is being brought up, DMKOPE reads the first warm start cylinder record to check if the automatic IPL bit is turned on. (The automatic IPL bit will be turned off in the warm start data after it is checked.) If it is on, and there is valid warm start data, no operator intervention is needed during IPL. The current TOD clock value is used and a warm start takes place. Note that an automatic IPL can take place only if the SHUTDOWN POWEROFF command was issued, warm start data was saved, and the TOD clock on the processor has been set.

The Auto Start feature is supported only for 4361 processors. Using a battery operated clock, the feature maintains the time-of-day (TOD) clock while the power is off and sets the TOD clock after power on.

Dumping the System and Automatic Re-IPL

When a system abend occurs or when the system console restart button is pressed, the module DMKDMPDK dumps all of main storage or just the CP portion of main storage to the indicated dump device. After the dump is completed, DMKDMPRS issues an IPL CCW to read the IPL sequence and, subsequently, DMKCKP from the SYSRES just as during system shutdown. DMKDMPRS then loads the initial PSW from location 0 of main storage and DMKCKPT gets control. If the dump was to a printer or to a tape, then DMKDMP leaves CPID set to "CPCP" and DMKCKPT will conduct an orderly system shutdown just as if the SHUTDOWN command had been issued. If the dump was to a DASD, then DMKDMP sets CPID to "WARM" before loading DMKCKP. DMKCKPT will still perform an orderly system shutdown and then an automatic warm start will be performed just as if the SHUTDOWN REIPL command had been issued.

Attaching a Virtual Machine to the System

After CP has been initialized, DMKCPVEN enables the communication lines in response to the ENABLE command. Then an individual virtual machine is attached to the system, using the following steps[3]:

1. *Terminal Identification*

   When the CP receives the initial interrupt from a terminal on an enabled line (normally initiated by a user dialing in on a data-set), the DMKCNSIN routine is entered. DMKCNSIN determines the terminal device type, stores this information in the terminal device block, writes the online message and puts the terminal line in a state to receive an attention interruption.

2. *Attention from User*

   After the online message has been displayed at the user's terminal, and he has pressed the ATTENTION key, DMKCNSIN (the console interruption routine) calls DMKBLDVM to build a skeleton VMBLOK for the user. At this time, the userid is LOGONxxx, where xxx is the terminal real device address, and a flag is set to indicate that the user has not yet completed the logon process.

   Then DMKCNSIN calls DMKCFMBK, which types a single blank at the terminal, and issues a read to the terminal for the user to enter his first command (normally LOGON or DIAL).

3. *First Command from User*

   After the first command has been entered by the user, DMKCNSIN further determines the type of terminal. If the terminal is a 2741, DMKTRMID is called to identify it as either a 2741P (PTTC/EBCD) or a 2741C (Correspondence) terminal. If successful, the correct device type

[3] The process described here ignores VSCS terminal attachment support.

and translate tables for input and output are set; if not, flags are set to indicate that the terminal is not yet identified.

Then control is returned to DMKCFMBK, which determines if the first command is valid (for example, LOGON, MSG, or DIAL). If the first command is not valid, a restart message is given, and the read to the terminal occurs again for the first command. If the first command was LOGON (or its abbreviation), DMKLOGON is called to complete the process of attaching the virtual machine to the system.

The operations performed by DMKLOGON include the following:

- Obtains the userid from the command line, and checks for a possible password and other optional operands
- Checks the userid and password against entries in CP's directory of users
- Ensures that the user is not logged on at another terminal (an error condition), or reconnects the user if he was running in disconnect mode
- Obtains pertinent information on the user's virtual machine from the user machine block portion of the directory
- Stores the correct userid (replacing the LOGONxxx userid used until now), virtual storage size, and other vital information in the virtual machine's VMBLOK
- Allocates and initializes segment, page, and swap tables (necessary for handling of the virtual machine's virtual storage)
- Schedules MSS volume mounts for any required MSS volumes if the MSS is available and the volume is not already mounted
- Allocates an extended VMBLOK (ECBLOK) if the user's virtual machine has the ability to run in the extended control mode
- Allocates and initializes virtual device blocks, control unit blocks, and channel blocks, using information from the user device blocks portion of the directory
- Establishes links (as feasible) to all DASD included in the directory, the accessibility of any disk being determined by the user access mode in the directory, and whether any other users are presently linked to the disk, in read mode and/or write mode
- Initializes all other virtual device blocks as appropriate, such as reader, punch, printer, and terminal
- Maps all virtual devices to real devices
- Performs appropriate accounting

- Informs the user of the date and time of the most recent revision to the system log message (LOGMSG), and of the presence of any outstanding spooled files in his virtual reader, printer, or punch

- Sends a ready message to the user with the date and time (and weekday), and a message to the system operator indicating that the user has logged on.

If the virtual machine has a device address or a named system in the directory and the initialization was not suppressed via an option on the LOGON command line, then that device or named system is then loaded (via IPL) at the conclusion of the logon process. Otherwise, when the logon functions are complete, the user's terminal is placed in CP read mode ready for the entry of his first desired command.

Under the latter condition of no automatic IPL, the user can IPL an alternate nucleus by using the STOP option in the IPL command. This option causes the normal IPL procedure to halt execution prior to loading the initial PSW, and issues a DIAGNOSE code 8 that place the user's terminal in CP read mode. A hexadecimal character entered in location X'08' changes the nucleus name. A hexadecimal character entered in location X'09' changes the apparent storage size. The BEGIN command allows the IPL procedure to continue.

System Reconfiguration

The 3081 Processor Complex uses the 3082 Processor Controller to coordinate central communications for the processor complex. The Monitoring and Service Support Facility (MSSF) is a hardware component of the processor controller. MSSF supplies I/O configuration and storage information for the 3081 Processor Complex and executes commands that modify the real system configuration.

VM/SP uses the MSSFCALL instruction to communicate with the MSSF. MSSFCALL is a diagnose instruction with a function code of X'80'. A command request to the MSSF requires a hardware call control block (HCBLOK). CP modules use the HCBLOK to issue an MSSFCALL instruction. The HCBLOK is a 48 byte storage area that is used to order requests to the MSSF and provide status of the requests to the caller.

After executing a command, MSSF uses the MSSF data block (MSFBLOK) to return information to the requester. MSFBLOK is a 2K byte storage area that MSSF uses to return header information and other command dependent information. The maximum length of the MSFBLOK is 2K; however, CP always obtains a page of storage to ensure that the control block is on a 2K storage boundary. The page of storage is locked to prevent page faults during processing of the request. For a description of the fields in the HCBLOK and the MSFBLOK, refer to *VM/SP Data Areas and Control Block Logic Volume 1 (CP).*

MSSF provides real processing for VARY PROCESSOR and SCPINFO commands; however, if the VLOG option of the VARY OFFLINE PROC command is used, the processor is switched offline to VM/SP. MSSF is not called to physically switch the processor offline. For real MSSF processing, the SCPINFO command must originate from a V=R virtual machine.

Real MSSF Processing - VARY PROCESSOR

Module DMKCPU receives control to execute a VARY PROC command. If the VARY PROCESSOR command is issued on a 3081 processor (CPUID = 3081 in PSA), DMKCPU calls DMKPTR to obtain a page of storage for the HCBLOK and MSFBLOK; the HCBLOK uses the first 2K of storage while the MSFBLOK uses the remaining 2K. DMKCPU sets the HCBLOK fields to contain the address of the MSFBLOK, the MSSF command word, and the return address for MSSF interruptions. DMKCPU passes control to DMKMHC (at entry point DMKMHCCP) to schedule the MSSF request.

Real MSSF Processing - SCPINFO and IOCP

Module DMKHVC receives control to validate the diagnose instruction produced by an IOCP or SCPINFO command. DMKHVC calls DMKMHV to process the diagnose instruction. DMKMHV examines the MSSF external interruption pending field (VMMSSFXP) in the virtual machine's VMBLOK to determine whether the virtual machine has an outstanding MSSF request. If an MSSF external interruption is pending, the virtual machine's PSW (VMPSW) is set to condition code two (MSSF busy), and control returns to DMKHVC. If a request is not pending, DMKMHV does the following:

- Sets VMPSW to indicate condition code zero
- Sets VMMSSFXP in the VMBLOK for the virtual machine
- Builds an HCBLOK
- Sets the HCVMREQ field to indicate a virtual machine generated MSSFCALL operation
- Passes control to DMKMHC (at entry point DMKMHCVM) to schedule the MSSF request.

If DMKMHV is processing the IOCP command, before passing control to DMKMHC, DMKMHV verifies that the processor is a 3081 and examines the virtual machine user's privilege class. The virtual machine user must have privilege class C or E to read from an input/output configuration data set (IOCDS) and privilege class C in order to write to an IOCDS. If the user does not have the appropriate privilege class, DMKMHV sets the invalid command response code 01F0 and returns control to DMKHVC.

For an IOCP write request, DMKMHV limits access to an IOCDS to one user at a time. DMKMHV obtains a lock to serialize access to an IOCDS. Because DMKMHV is pageable, the IOCP write lock (ICPWLOK) is defined in module DMKMHC at entry point DMKMHCLK. DMKMHV uses an external reference to DMKMHC to examine the lock.

- If the lock is free, DMKMHV ensures that the IOCDS is open and sets the lock for the virtual machine user.
- If the lock is held, DMKMHV determines whether the lock is held by this user. If the lock is not held by this user, DMKMHV sets condition code two (MSSF busy) and returns control to DMKHVC.
- If DMKMHV determines that the lock is held by this virtual machine user, DMKMHV prepares to perform a real MSSF open for the IOCDS.

After obtaining the lock, DMKMHV builds an HCBLOK to contain the address of the MSFBLOK and MSSF command word, and calls DMKMHC to schedule the request. DMKMHV calls DMKMHC to schedule the request. The lock is not needed to process an IOCP read request. DMKMHCVM is the entry point for MSSFCALL instructions generated by a virtual machine.

Scheduling and Executing the Real MSSF Request

Module DMKMHC schedules all CP and virtual machine requests for MSSF operations. DMKMHC contains several entry points. DMKMHCCP is the entry point for CP generated MSSFCALL operations while DMKMHCVM is the entry point for virtual machine MSSFCALL requests. Though the entry points are different for the operations, processing is similar. On entry, DMKMHC obtains the address of the HCBLOK and performs the following functions:

- Verifies that the processor-id is that of a 3081 processor
- Sets the HCBLOK pointer, flag bytes, and priority fields to zeros
- Sets the HCMSFSYS field to indicate a CP or virtual machine generated MSSFCALL operation
- Saves the system VMBLOK address in the HCUSER field of the HCBLOK
- Adds the address of the HCBLOK to the internal control block pointer chain (HCANCHOR)
- Examines the active pointer (HCACTIVE) to determine whether MSSF is busy with a previous request.

HCACTIVE contains the address of the HCBLOK that MSSF is currently processing. The HCACTIVE field is zero if the MSSF is available. If the MSSF is available, DMKMHC passes control to subroutine MSFCALL (using BALR) to issue the diagnose instruction. Subroutine MSFCALL performs the following:

- Obtains the address of the HCBLOK from the queue
- Issues the MSSFCALL diagnose instruction
- Traces the diagnose instruction (if internal tracing is active)
- Validates the condition code returned by MSSF.

If MSSF returns condition code zero, the address of the HCBLOK is placed in HCACTIVE and HCFLAG is set to indicate that MSSF is processing the diagnose instruction. The HCBLOK is deleted from the queue of pending requests and control is returned to the caller.

The MSSFCALL instruction and MSSF response are traced if the internal trace facility is active. For further information on the MSSFCALL diagnose instruction see *VM System Facilities for Programming.* Refer to *VM Diagnosis Guide* for the format of the trace table entries.

Processing the MSSF Interruption

MSSF uses an external interruption (which was enabled in PSW bit 7 and control register 0 bit 22) to signal CP that processing is complete. MSSF issues the interruption code X'2401' to the CP external interruption handler (DMKPSA) and stores the absolute address of the MSFBLOK in the INMSFBLK field of the PSA. DMKPSA passes control to DMKMHC. DMKMHC processes external interruptions resulting from the MSSFCALL instruction at entry point DMKMHCIN.

At entry point DMKMHCIN processing is as follows:

- Obtains the global system lock

- Verifies that the interruption from DMKPSA is associated with the HCBLOK address in the HCACTIVE field

- Examines the interruption to determine whether CP or a virtual machine initiated the request.

For a CP request, DMKMHC obtains a CPEXBLOK and initializes it to contain the interruption return address (DMKCPUMI) and the address of the MSFBLOK. DMKMHC places the CPEXBLOK on the dispatcher queue and control returns to DMKCPU. DMKCPU examines the MSSF completion code and issues appropriate messages.

For a virtual machine request (HCVMREQ = 1), processing is as follows:

- Resets the interruption pending field (VMMSSFXP) in the virtual machine's VMBLOK

- Obtains an XINTBLOK for the virtual machine

- Determines whether the virtual machine is logging off or performing a system reset

- Releases the HCBLOK and MSFBLOK

- Places the XINTBLOK on the virtual machine's external interruption queue

- Exits to the dispatcher to return control to DMKHVC.

DMKMHC compares the HCACTIVE field with the address of the MSFBLOK. If the addresses do not match, ABEND MSF02 is generated. If they match, DMKMHC uses zeros to clear HCACTIVE and examines the queue of pending MSSF requests. If another request is in the queue, DMKMHC sets HCACTIVE to the address of the new HCBLOK, and the new request is executed.

When a virtual machine logs off or performs a system reset, DMKCFP receives control. DMKCFP (at entry point DMKCFPRR), calls DMKMHC to determine whether the virtual machine has an outstanding MSSF request. At entry point DMKMHCRE, processing is as follows:

- Clears VMMSSFXP in the virtual machine's VMBLOK
- Examines the queue of pending MSSF requests
- Deletes the HCBLOK from the queue if the request is not active
- Releases storage for the MSFBLOK and the HCBLOK
- Examines the IOCP write lock
- Clears the lock if it set for this virtual machine
- Returns control to DMKCFP through the dispatcher.

If the virtual machine has an active MSSF operation, DMKMHC waits for the external interruption before releasing the MSFBLOK and HCBLOK.

Virtual MSSF Processing - SCPINFO

Module DMKHVC receives control to validate the diagnose instruction produced by a virtual machine SCPINFO command. DMKHVC calls DMKMHV to process the diagnose instruction. DMKMHV simulates the MSSFCALL instruction by setting the virtual machine's PSW to contain a condition code and scheduling external interruptions in the same sequence as would be presented on the real processor.

DMKMHV examines the XINTBLOKs for the virtual machine to determine whether the virtual machine has an outstanding MSSF request. If an MSSF external interruption is pending, the virtual machine's PSW (VMPSW) is set to condition code two, and control returns to DMKHVC. If there is no MSSF request pending for the virtual machine, VMPSW is set to condition code one. DMKMHV then verifies the MSFBLOK. If there are no violations, the MSFBLOK is set up to contain predefined response codes:

After setting the MSFBLOK as shown, DMKMHV obtains an XINTBLOK for the virtual machine and sets it to the address of the MSFBLOK. The XINTBLOK is placed on the virtual machine's external interruption queue and control returns to DMKHVC through the dispatcher. Control returns to DMKHVC.

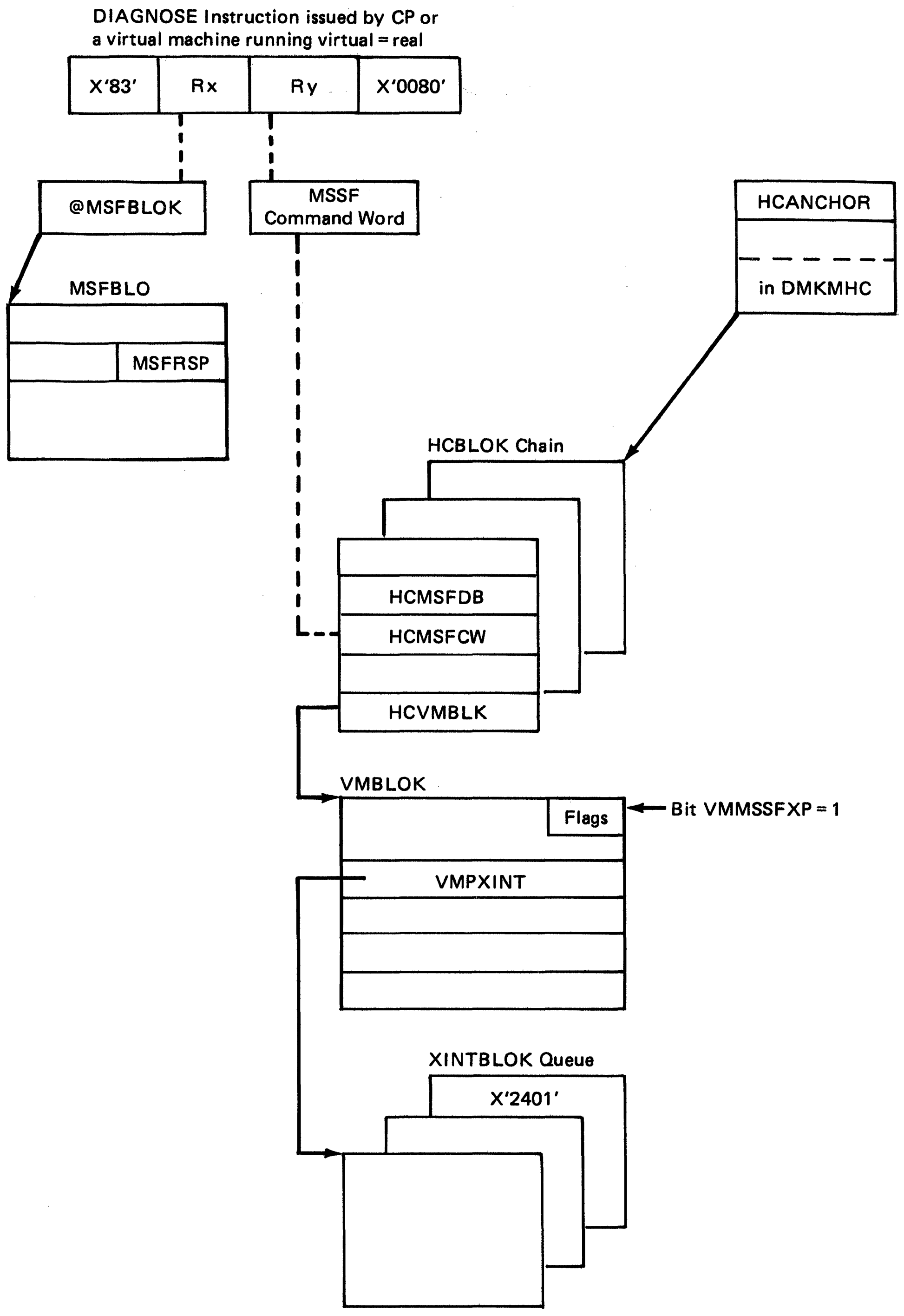


Figure 27. Real MSSFCALL Control Block Structure

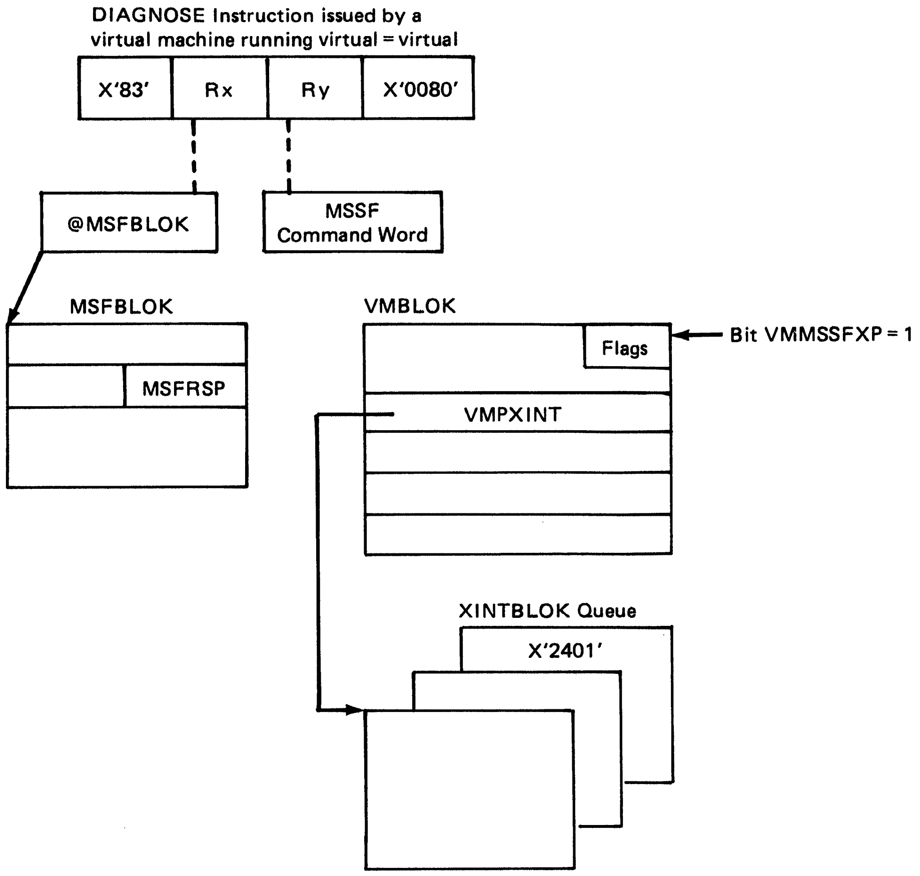


Figure 28. Virtual MSSFCALL Control Block Structure

I/O Reconfiguration

Three commands alter the I/O configuration of a user's virtual machine after he has logged on. Two are user commands, while the third a system operator command, because it affects the status of real devices attached to the system. The ATTACH and DETACH commands are contained in DMKVDA, DMKVDC, DMKVDD, and DMKVDE and the DEFINE command in DMKDEF. The system command scanner (DMKCFM) calls both pageable modules after their format and privilege classes have been validated. These commands access the same control-block building subroutines in the module DMKVDS that DMKLOG, the LOGON processor, uses.

Attaching a Real Device: The system operator can dedicate any real device to a single virtual machine by issuing the ATTACH command. The device attached is available only to the given virtual machine, and all I/O requests to it are handled by CCW translation. If the device is a DASD, cylinder relocation does not occur when SEEK addresses or home addresses are referenced. The I/O supervisor does not queue operations on the device, nor does it automatically restart it or do ordered seek queuing. Nonsharable devices such as tape drives must be attached to a virtual machine to be accessed by the virtual machine. A virtual machine can also have a dedicated card reader/punch or printer. However, this is usually not necessary because of the unit record spooling facilities of CP. Unit record input or output on a dedicated (attached) device is not spooled by CP. The unit attached may be given a virtual address different from its real address; however, the virtual machine may not already have a virtual device at the attached address. A real device cannot be attached:

- If it is currently dedicated to another virtual machine, or
- If it contains minidisks that are in use by other virtual machines, or
- If it is a system-owned volume that is in use for spooling or paging.

In addition, the device must be attached at an address where the VCU (if one exists) is using the matching subchannel protocol. For example, if you have a 3420 tape drive (which uses shared protocol) at virtual address 181, you cannot attach a 3211 printer (which uses nonshared protocol) at address 182.

The system operator can dedicate a remote 3270 Information Display System Printer (3284, 3286, 3287, 3288, or 3289) to a single virtual machine by issuing the NETWORK ATTACH command. The printer attached is available only to the given virtual machine, and all I/O requests to it are handled by Start I/O.

Defining a Virtual Device: A system user can define a new virtual device with the DEFINE command that does not require the dedication of a corresponding real device. Devices that can be defined are consoles, spooled readers, punches and printers, dialable TP lines, virtual channel-to-channel devices, pseudo timers, and temporary disks. With the DEFINE command, the user can change any existing virtual device address whether it corresponds to a shared or dedicated real device or no real device unit.

Care must be taken when using the DEFINE statement to create (or move) a virtual device. CP checks for a potential subchannel protocol conflict on the virtual control unit that would support the new device. If a conflict is detected, the operation is not performed. Instead, CP sends an error message to the user who issued the command.

The DEFINE command can also describe the virtual machine channel mode of operation, that is, either selector or block multiplexer. The default mode, selector channel mode, reflects a channel busy to any SIO operation attempted on the same channel path that has not completed the previous channel SIO operation. Block multiplexer mode allows the successful initiation of different devices on the same channel path. Channel 0, a byte-multiplexer channel, is unaffected by the DEFINE command. Use of the DEFINE command with the CHANNELS operand generates a virtual machine reset; therefore, it should be invoked prior to the virtual machine IPL operation.

Note: The channel mode selected has no bearing on the types of channels that are attached to the real system.

Temporary disks are dynamically obtained cylinders or blocks of DASD storage space. They are available to the user for as long as they are part of his virtual machine configuration, but the data on them is destroyed after the user detaches the area. For all other purposes, however, they appear to be a standard disk.

Detaching a Virtual Device: A virtual device can be removed from a virtual machine configuration prior to logging off with the DETACH command. A user can detach any of his own devices, and the system operator can detach a real device from a virtual machine. If the operator detaches the device, the user is informed of the operator's action. A real device can be detached only if it is dedicated to a single virtual machine or is attached to the system and is not in use when the DETACH is issued.

The system operator can detach a remote 3270 Information Display System Printer (3284, 3286, 3287, 3288, or 3289) from a virtual machine with the NETWORK DETACH command. If the operator detaches a 3270 Information Display System Printer, the user is informed of the operator's action. A 3270 Information Display System Printer can be detached only if it is dedicated to a single virtual machine or is attached to the system and is not in use when the NETWORK DETACH command is issued.

Disconnecting a Terminal or Virtual Machine

A user may permanently or temporarily disconnect his terminal or virtual machine from the system by a console command, or the terminal or virtual machine may be forcibly disconnected by the operator. The system can also log off the virtual machine. In any case, the routines that handle the termination process are in the pageable module, DMKUSO.

Permanent Disconnect: The user may voluntarily remove his virtual machine from the system via the LOGOFF command. This command terminates all virtual machine operation, releases all storage occupied by control blocks and virtual storage pages, and disconnects the teleprocessing line connection to the user's terminal. If the user specifies the HOLD option with LOGOFF, all of the above occurs, except that the teleprocessing line remains enabled. This option is especially useful for dialed connections that are reused immediately by another user.

The virtual machine can be forced off the system by the system operator via the FORCE command. This has the same effect as a user-initiated logoff, except that the user is not informed that the operator has logged off his machine. A virtual machine may also be logged off the system if the virtual machine is running disconnected (without an active terminal) and the virtual machine attempts a terminal read or enters a disabled wait state.

The DMKUSOLG and DMKUSOFF subroutines process the LOGOFF command. DMKUSOFL processes the FORCE command.

Temporary Disconnect: A user may temporarily disconnect his terminal from his virtual machine by using the DISCONN command, while allowing the virtual machine to continue to run. This command flags the virtual machine as being disconnected and releases the user's terminal and teleprocessing line. If the HOLD option was specified in the DISCONN command, CP allows the line to remain enabled, and another user can use the terminal to log on. The disconnected virtual machine continues to be dispatched until it either attempts to execute a terminal read to the disconnected console or it enters a disabled wait state. At this time, the dispatcher (DMKDSP) calls the routine DMKUSOFF directly to force the machine out of the system. While the machine is disconnected from its virtual console (real terminal) any terminal output is lost; in addition, CP may apply a disconnected penalty to the machines scheduling priority, to bias the system in favor of interactive users.

A user's virtual machine may also be disconnected by the system. If the disconnected user logs on to the system while the disconnected machine is still running, it is reconnected and can continue to interact with the system in the usual manner.

The DMKUSO subroutine processes the DISCONN command.

Console Functions

DMKCFM analyzes CP commands and passes control to the appropriate routine to handle the command. DMKCFM can be entered by the Attention key (or equivalent) at the user's terminal or directly from a virtual machine.

When a console interruption occurs by the Attention key at the user's terminal, DMKIOSIN calls DMKCNSIN to handle the unsolicited interruption, then DMKCNSIN calls DMKCFMBK.

DMKCFMBK first calls DMKFREE to obtain storage for an 18-doubleword input buffer. Next, DMKQCNWT is called to send the CP message to the terminal to inform the user that he has entered console function mode. DMKQCORD is then called to read the command line entered at the console.

DMKCFMEN is the entry point for commands coming directly from the virtual machine. DMKPRGIN enters at DMKCFMEN here when a DIAGNOSE instruction with a code of 8 is detected. The address of an 18-doubleword input buffer is passed in register 1; therefore, a read to the terminal is not needed.

After either the read to the terminal or entry from the virtual machine, DMKSCNFD is called to find the command type. On return from DMKSCNFD, register 1 points to the start of the command and register 0 contains the length of the command. DMKCFM then calls DMKCFCMD to scan and validate the command. DMKCFCMD contains a command table which it searches until it finds a command name match. It compares the user classes in the field CTCLASS. If the user is authorized to issue the command, DMKCFCMD will pass the appropriate module entry point from the command table to DMKCFM. The command is then processed. For the commands ATTACH, DETACH, INDICATE, NETWORK, QUERY, and SET, DMKCFCMD uses subcommand tables in DMKCMD to locate the Appropriate module entry points.

If DMKCFCMD cannot find a command name match, the user receives a message informing him that the command or subcommand table search has failed.

After the command has been processed, control is returned to DMKCFM. There are three possible returns:

- On a normal return, the input buffer is scanned to see if there are any more commands. If none exist, DMKCFM returns to the virtual machine (if entered via DIAGNOSE) or calls DMKQCORD to read the next command from the terminal.
- On a return plus 4, the VMCFWAIT bit is turned off to allow the virtual machine to run. DMKFRET is called to return the input buffer storage. Then control returns to either the virtual machine, if entered via a DIAGNOSE, or to DMKDSPCH if entered via the Attention key.
- On a return plus 8, the operation is the same as plus 4 except that the VMCFWAIT bit is left on.

Dispatching and Scheduling

The scheduler, DMKSCH, selects dispatchable virtual machines from the virtual machine population. The auxiliary routine that assists the scheduler and dispatcher is the request stack maintenance routine, DMKSTK.

To make decisions on dispatching and scheduling, the control program places all virtual machines into various categories, and recognizes user machines as being in one of several states. The virtual machine categories, either interactive or noninteractive, are defined in the following way:

- An interactive virtual machine is one whose use of the system is punctuated by regular and frequent terminal I/O, and does not have long processor execution times. A virtual machine becomes eligible to enter interactive status whenever a channel program for virtual console I/O has completed, or whenever I/O for a dedicated or dialed virtual telecommunications line has completed.

- A non-interactive virtual machine is one that has violated an interactive criterion, or one that has entered an idle wait state by entering console function mode (equivalent to stopped state), or by loading a wait state PSW that is not enabled for any busy channel. CP schedules interactive users ahead of non-interactive users. Non-interactive users are subdivided into several classes. Normal non-interactive virtual machines are scheduled by a priority scheme described below. A virtual machine is allowed to execute for a specified time period and then it is placed in a list of those machines that are waiting.

Furthermore, virtual machines are categorized on a sliding scale by their levels of resource utilization. The actual scheduling sequence depends on both a virtual machine's interactive/noninteractive characteristics and its current resource utilization level.

To give preference to certain classes of virtual machines, a priority scheduling scheme allows virtual machines to be scheduled with a priority class. The priority is a number assigned by the directory; however, the number may be altered by the system operator. This priority is referred to as the `user priority` to distinguish it from the priority used by the scheduler for ordering virtual machines for service.

Virtual Machine Dispatching Lists and States

To efficiently manage the large inventory of potential virtual machines that are logged on to the system, CP defines several states that a virtual machine may occupy. The scheduler can move a virtual machine from one state to another; however, a virtual machine may exist in only one state at any given instant. CP can then make scheduling and dispatching decisions by looking only at the subset of virtual machines that are in the appropriate state. To do this search, it also maintains lists of virtual machines in certain executable states.

A user's virtual machine may be in one of the following states:

State	Meaning
1	Interactive and dispatchable (in queue1, in dispatch list)
2	Interactive and not dispatchable (in queue1, in dispatch list)
3	Interactive and eligible for queue1, but no available real storage (waiting for queue1, in eligible list)
4	In wait state with terminal read or write active
5	Non-interactive and dispatchable (in queue2, in dispatch list)
6	Non-interactive and not dispatchable (in queue2, in dispatch list)
7	Non-interactive and eligible for queue2, but no available real storage (waiting for queue2, in eligible list)
8	Idle - waiting for asynchronous I/O or external interruption, or stopped (in console function mode)

Entries on the dispatch list are the VMBLOKs for those virtual machines in states 1, 2, 5, and 6, and represent the virtual machines that can be run at any given time. (States 2 and 6 remain in the dispatch list even though they are not dispatchable.) The dispatch list is sorted by deadline priority, which is the time of day (TOD) by which the virtual machine should have used its allocated processor time and been dropped from queue. A task is defined as that execution that takes place between terminal reads or entry to enabled wait (that is, movement from state 4 or 8 to state 1) and is re-projected for a virtual machine each time it is dropped from a queue. The deadline priority is recalculated each time a virtual machine is dropped from queue and represents the next expected time that the virtual machine should again be dropped from queue.

The eligible list contains virtual machines in states 3 and 7. These virtual machines are potentially executable, but due to the current load on the system they are not allowed to compete for the processor. As soon as a virtual machine in the dispatch list is dropped from queue, the highest-priority virtual machine(s) in the eligible list is added to the dispatch list. Conditions can arise where the virtual machine that is added to the dispatch list has a projected working set size that far exceeds the remaining system capacity. The eligible list has two components; a section composed of those virtual machines waiting for Q1 (interactive) and a section composed of those virtual machines waiting for Q2 and Q3 (non-interactive). Each section of the list is sorted by deadline priority, which is determined at the time the virtual machine was previously dropped from queue, as follows:

- Paging ratio - The virtual machine's projected working set size, calculated the last time it was dropped from a queue, is expressed as a ratio to the overall system average working set size.

- Processor resource ratio - The virtual machine's processor resource utilization the last time it was dropped from a queue, is expressed as a ratio to the overall system average processor resource utilization.

- Average queue delay - The total eligible list time divided by the total list time (sum of the total eligible and dispatch list times) for all virtual machines.

- User bias ratio - A user bias ratio is formed by multiplying the paging ratio by the paging bias and adding that to the result of multiplying the processor resource ratio by 100 minus the paging bias. The resulting sum is divided by 100 to arrive at a user bias. In effect, this is a weighted average of the paging and processor resource ratios, where the weights are determined by the paging bias. The higher (lower) the real storage contention, the larger (smaller) the weight given to the paging ratio.

- Prioritized Q2 delay factor - The Q2 delay factor, which is the sum of the average times a virtual machine spends on the eligible list and on the dispatch list, is multiplied by the function (64**(UP/64))/64, where UP is the user priority. For those priorities equal to 64, the function is 1, for those greater than 64, the function is greater than 1, and for those less than 64, the function is less than 1.

- Virtual machine queue delay factor - The user bias ratio is multiplied by the user-prioritized Q2 delay factor to arrive at the virtual machine's queue delay factor. The queue delay factor is stored in the VMBLOK in the VMQPRIOR field. The units of the queue delay factor are equal to the high-order word of the time-of-day clock multiplied by 32 (approximately 1/32 of a second).

- Q3 - If a given virtual machine is (a) a Q2 virtual machine, and (b) it is dropping from Q2 because it has used up its allocated processor in-queue time, and (c) it has dropped from Q2 at least 8 consecutive times because of using all allocated processor resources (without entering Q1), then the virtual machine is marked as Q3 and the queue delay factor is multiplied by 8. A Q3 virtual machine is treated as a normal Q2 virtual machine, except for differences in deadline priority calculation and the amount of processor time the virtual machine is allowed in queue.

- Virtual machine deadline priority - The virtual machine's queue delay factor from Step 6 is added to the time of day when the virtual machine drops from queue. This value becomes the virtual machine's deadline priority and represents the expected time of day that the virtual machine should be next dropping from queue. The deadline priority is stored in the VMBLOK in the VMEPRIOR field.

- Q1 deadline priority - If the virtual machine is being added to the Q1 eligible list, then the deadline priority is adjusted to account for the shorter processor time allowed in Q1. The queue delay factor (from VMQPRIOR) is subtracted from the deadline priority (from VMEPRIOR). The queue delay factor is divided by 8 (shifted right 3) and stored back in VMQPRIOR as the new queue delay factor. In addition, if the virtual machine is currently using less than its allocated processor resource, the queue delay factor is shifted right additionally the number of places determined by the interactive bias. In the standard system the interactive bias is set to 2. The new queue delay factor is then added back into the result of the subtraction to form the Q1 deadline priority, which is then stored into VMEPRIOR.

The VMBLOK is then sorted into the appropriate section of the eligible list in ascending value of VMEPRIOR. The units used in the deadline priority calculation are obtained by taking the high-order word of the time-of-day clock and shifting it left 5 bit positions. The low-order bit position is approximately equal to 1/32 of a second. To handle the effects of losing the high 5 bits of the time-of-day clock, the current time-of-day value first has the time of day at system IPL subtracted from it before shifting. The effects of the individual virtual machine's resource utilization and user priority are illustrated by the following examples:

```
Example 1:  Assume that two virtual machines are to be added
to the eligible list for Q2.  The current paging bias
percentage is 0, both user priorities are 64, and the
current Q2 delay factor is 10 seconds.  Virtual machine A
has a current resource utilization (user bias) of 1/2 the
system average.  Virtual machine B has a resource
utilization equal to the system average.  The queue delay
factors are obtained as follows:

Step 1:

Virtual Paging           Processor            Combined
Machine Ratio              Ratio                Ratio
A        0 x ?    +    (100-0) x 1/2   =      50/100 = 1/2
B        0 x ?    +    (100-0) x 1     =      100/100 = 1

Step 2:

Virtual                Virtual Machine           Queue
Machine  Q2 delay      Priority ratio            delay
A         10      x      1             =          10
B         10      x      1             =          10

Step 3:
                                               Virtual
Virtual  Combined           Queue               Machine
Machine   ratio             delay                delay
A          1/2     x         10        =           5
B            1     x         10        =          10
```

If both virtual machines A and B dropped from queue at time-of-day 0, then virtual machine A has a deadline priority of 5 seconds and virtual machine B has a deadline priority of 10 seconds. Assuming virtual machine B continues to run, it should drop from queue once every 10 seconds, which is the system average. By cycling through queue at the average system rate, virtual machine B should maintain its processor resource utilization at the system average level (which will maintain its virtual machine delay at 10 seconds). Virtual machine A, whose current processor resource is 1/2 the system average, is allowed to cycle through queue in 1/2 the normal time. If virtual machine A continues to run at this rate, its processor resource utilization is accumulating at twice the system average. As virtual machine A's processor resource utilization approaches the system average, its combined ratio approaches 1. This increases its delay factor to 10 seconds, and its and virtual machine B running at the same rate.

Example 2: The conditions are the same as example 1, except both virtual machine A and B have the same processor resource utilization. Virtual machine A has a virtual machine priority of 64 and user B has a user priority of 54.

Step 1:

Virtual Machine	Paging Ratio		Processor Ratio		Combined Ratio
A	0 x ?	+	(100-0) x 1	=	100/100 = 1
B	0 x ?	+	(100-0) x 1	=	100/100 = 1

Step 2:

Virtual Machine	Q2 delay		Virtual Machine Priority ratio		Queue delay
A	10	x	1	=	10
B	10	x	1/2	=	5

Step 3:

Virtual Machine	Combined ratio		Queue delay		Virtual Machine delay
A	1	x	10	=	10
B	1	x	5	=	5

Virtual machine B continues to have queue delays calculated smaller than virtual machine A (or the system average) as long as its processor resource utilization is less than twice the system average.

The above examples illustrate the following general points about the deadline priority scheduling:

- The purpose of the calculations is to effectively control each virtual machine's resource utilization rate by controlling how fast each machine is able to move through queue each time.
- Each virtual machine's default allowed utilization is the same, namely the system `fair share` value, that is, the total system resources divided by the number of active virtual machines.
- Deviations from a virtual machine's specified resource utilization (because of any particular advantageous or disadvantageous stay in queue) are corrected for successive stays in queue.
- The system's response to any virtual machine's transaction is proportional to that virtual machine's current resource utilization and the total system load. Very trivial, interactive virtual machines receive the best service. Service for others degrades proportionally to their current level of utilization.

Paging Percentage Bias

The paging percentage bias is a dynamically calculated value that is proportional to the percentage of time spent in the eligible list. The upper limit of the paging percentage bias is 40 percent. The paging percentage bias represents the portion of the utilization ratio due to paging, rather than processor utilization. The maximum paging percentage bias can be queried and set with the QUERY SRM and SET SRM commands.

Interactive Bias

The interactive bias shift is set to 2. Normally when a virtual machine enters the Q1 eligible list, its priority delay factor is shifted right 3 (divided by 8). In addition, if the virtual machine's current resource utilization is less than its allocated amount (or fair share), the priority delay factor is shifted right an additional amount specified by the interactive bias (2). This allows much faster response for trivial interactive virtual machines, but still prevents the nontrivial interactive virtual machines from obtaining more than their fair share of the system resources. The interactive bias can be queried and set with the QUERY SRM and SET SRM commands.

User Priority Function

The user priority function is a nonlinear function, and changes to user priorities affect performance differently, depending on the original priority. The priority ratio is used to pro-rate a virtual machine's queue delay. The smaller the ratio (and the smaller the priority), the shorter the queue delay and therefore the better the service for that virtual machine. The following is a table of user priorities and the priority ratio calculated from the user priority function.

In-Queue	Not-in-Queue
0	1/64
11	1/32
21	1/16
32	1/8
43	1/4
54	1/2
64	1
75	2
85	4
96	8

A user priority of 0 provides a virtual machine with a `fair share` 64 times as large as the average (by multiplying its actual use by 1/64). A user priority of 121 provides a virtual machine with a `fair share` 1/32 of the average (by multiplying its actual use by 32). The normal priority is assumed to be 64 (which results in a multiplier of 1).

Controlling of Multiprogramming

To control the number of virtual machines allowed in queue, the scheduler monitors the paging activity of all virtual machines and of the total system. A decision as to whether or not to move a potential virtual machine from the eligible to the dispatch list is based upon whether or not that its projected working set exceeds the system's remaining capacity. A virtual machine's working set is calculated and projected at queue drop time. The estimate may also be pages it has resident.

Average Resident Pages: The basic mechanism used in predicting working set size is dependent upon the virtual machine's average resident pages while in queue. At each page fault, the current number of resident pages for that virtual machine is added to the resident page sum (VMPGRINQ). At queue drop the average number of resident pages for that virtual machine is approximated by dividing the resident page sum by the number of page faults.

Predicted Working Set Size: Before using as a working set size estimate, the average resident page value is adjusted upward or downward dependent upon how well the virtual machine (and overall system) ran while it was in queue.

There is a tuning parameter in the system, which is changeable by the CP SET PAGING command, that provides a `norm` with which to compare paging performance. This value is set at a default value of 8. It is used in two ways to adjust the average resident page value for use as the working set size estimate.

- The average resident page value is increased by 1 for each page steal in excess of 8 percent of the total number of page faults for that virtual machine while in queue.

- The average resident page value is multiplied by the square root of the virtual machine's paging performance ratio, which is defined as follows:

  At queue drop time, the estimated productive processor time per page fault is calculated. The total processor time used by the virtual machine while in queue is divided by the number of page faults it had while in queue. From this processor time per page fault value is subtracted the estimate processor time per page read. The scheduler maintains an `ideal` system-wide productive processor time per page fault value. Dividing the virtual machine's calculated productive processor time per page fault by the `ideal` system value gives the virtual machine's paging performance ratio for that time in-queue.

The adjustment formula for average resident pages is:

ARP' = (ARP + S) * (ICPU/PCPU)

where:

ARP'
: is the adjusted average resident page value.

ARP
: is the calculated average resident page value.

S
: is the number of page steals in excess of 8 percent of the number of page faults.

PCPU
: is the average productive processor time per page fault.

ICPU
: is the system's ideal productive processor time per page fault.

The predicted working set size estimate is set equal to the adjusted average resident pages (**ARP'**) and stored in the VMWSPROJ field in the VMBLOK. There are four constraints on the predicted working set size estimate:

- The minimum value for the predicted working set size is 5.
- The predicted working set size must be less than the total number of available system pages.
- The predicted working set size must be less than or equal to the larger of either:
  - The total number of page faults
  - The in-queue high-water mark for resident pages.
- The maximum value of a size is set by the SET SRM MAXWS command. Ordinarily this maximum is not active.

Assuming that page contention is being adequately controlled and the system resources have not been severely overcommitted, the predicted working set size would ordinarily be close to the number of average resident pages. However, this estimate is sensitive to the overall system paging activity for the following reasons:

- If there is no paging load on the system, the `ideal` productive processor time per page fault is set to its minimum value, and the working set sizes is continually underestimated.
- As paging activity increases, the ideal productive processor time is calculated larger and causes the square root calculation to approach 1.
- If the paging activity becomes excessive, and the system is unable to control it by reducing the multiprogramming level, the ideal productive

processor time figure approaches its maximum value. This normally happens when there is insufficient real storage or paging I/O capacity to meet the interactive (Q1) requirements.

In summary, the scheduler selects the subset of logged-on virtual machines that are allowed to compete for the resources of the processor. The scheduler prevents a virtual machine from being added to the active subset if the virtual machine's projected main storage requirements, added to those of the other active virtual machines, cause the current capacity of the system to be exceeded. Selection within scheduling priority means that an executable virtual machine of high priority is always added to the active subset (to a queue) before an executable virtual machine of lower priority. Once the active subset (the set of in-queue virtual machines) has been selected, the dispatcher allocates resources of the processor among them.

The list of executable virtual machines in a queue (on the dispatch list) is sorted by deadline priority (the same as for the eligible list). The only exceptions to this rule are machines identified as being compute-bound, that is, which use at least the dispatching time-slice amount of processor time without becoming nonrunnable. The dispatching time-slice is set at 50 ms for a System/370 Model 145 and is adjusted for other models based on the ratio of their speed to the Model 145 speed. In the case of a compute-bound virtual machine, the following value is added to the deadline priority for use in ordering the virtual machine on the dispatch list:

```
deadline TOD - current TOD
--------------------------
            4
```

The current TOD is taken as the time it was first determined that the virtual machine was compute-bound.

Virtual machines identified as interactive (in Q1) are not necessarily placed at the top of the dispatch list, although you can ordinarily expect to find them there. The virtual machine that is found at the top of the list is ordinarily one that:

- Has been waiting the longest since it was dropped from queue
- Has the smallest current resource utilization.

Normally, interactive virtual machines satisfy both of these requirements.

Working Set Size Estimate Feedback Control

A performance monitoring routine is invoked at intervals between 15 and 60 seconds (the interval depends on the processor speed and the multiprogramming level). This routine calculates among other things:

- The smoothed activity values used in the INDICATE LOAD command response

- The average Q2 delay

- The average processor resource utilization

- The `ideal` processor per page read.

Part of the response to the INDICATE LOAD command is a value that estimates the paging load on the system due to page contention, expressed as a percent of total system resources. The difference between this percent number and the SET PAGING value is divided by the number of page reads during the interval. This percent (positive or negative) is multiplied by the measurement interval and the resulting product is added to the `ideal` processor time per page fault.

The `ideal` processor time per page fault should control page contention by controlling the working set size estimates. Page contention can be reduced by reducing the level of multiprogramming. High levels of page contention (with respect to the SET PAGING value) cause the `ideal` processor time per page fault to increase. Eventually, increases in this `ideal` value cause all working set size estimates to increase, which should lead to decreases in the average multiprogramming level. The reverse situation, where page contention is lower than the SET PAGING value, causes a reduction in the `ideal` value, leading to smaller working set size estimates and possibly higher levels of multiprogramming.

Fast Redispatch

DMKDSP also provides a fast dispatch path for virtual machines that have issued specific privileged instructions that are not handled by virtual machine assist.

These virtual machines can be dispatched very rapidly because the virtual machine's program old PSW needs very little reconstruction to redispatch the virtual machine, hence use of full PSW reconstruction path is not required. The decision for using the fast dispatch path (DMKDSPA) is accomplished by the module that handles privileged operation, DMKPRV or DMKVIO. A fast redispatch path is also available after I/O interrupts. If DMKDSP can determine that the I/O interrupt processing had no effect on the running virtual machine's status and it caused no higher-priority virtual machine to become runnable, then the virtual PSW stored at the I/O old PSW location will be used to redispatch the virtual machine.

Enable Window

The CP supervisor runs disabled for all I/O and external interrupts. The dispatcher, in order to alleviate part of this problem, will temporarily enable for interrupts and then disable. There are three occasions when the dispatcher enables for interruptions (enable windows):

- When an enabled wait state is entered
- When an enabled problem state is entered to run a virtual machine
- When another part of the supervisor is to be entered via the unstacking of an CPEXBLOK.

On occasion 3, if the dispatcher finds a CP request block to unstack, it first enables, then disables for I/O and external interruptions before unstacking the request.

Favored Execution Options

When the resources of the processor (and real storage) are being allocated, the dispatching and scheduling functions are implemented in such a manner that options exist which allow an installation to designate that certain virtual machines are to receive preferential treatment.

The favored execution options allow an installation to modify the algorithms described above and force the system to devote more of (or a specific portion of) its resources to a given virtual machine than would ordinarily be the case. The options provided are:

- The favored execution option
- The favored execution percentage.

The favored execution option means that the virtual machine so designated is never to be dropped from the active (in-queue) subset by the scheduler. 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 option. An implicit commitment exists if neither of these options are 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 due to thrashing.

The basic favored execution option removes the primary source of elapsed time stretch-out in a loaded time-sharing environment. However, if the favored task is highly compute-bound and must compete for the processor with many other tasks of the same type, an installation can define the processor allocation to be made. In this case, the favored execution percentage option can be selected for the virtual machine. This option specifies that the selected virtual machines are to receive a given minimum percentage of the total processor time, if it can use it. It modifies the deadline priority to specify a fixed interval between queue drops instead of one proportional to system load, number of active virtual machines, current resource utilization, etc. The deadline priority is modified in the following way:

- The in-queue time slice is divided by the requested percentage to arrive at the fixed stretch-out interval.

- The calculated interval is added to the TOD at queue drop to arrive at the next expected queue drop TOD (or deadline).

These options can impact the response time of other virtual machines. Both favored options can be applied to any number of virtual machines, either in conjunction with one another or separately.

Dispatching and Scheduling Support Routines

Most of the routines in the CP nucleus are reenterable and multiple control program or virtual machine tasks can make use of one routine at the same time. However, there are certain areas where requests for a resource must be serialized (as in paging) or delayed while previous requests are serviced (as in requests to schedule I/O).

The CP Request Stack

The routine handling the request obtains a CPEXBLOK from free storage and stores the caller's registers in it; when the requested resource is free, the CPEXBLOK is stacked for the dispatcher via a call to the request stack manager (DMKSTK). The dispatcher unstacks the block and exits to the requesting routine the next time it is entered. I/O requests are stacked in the same manner, except that the stacking vehicle is the IOBLOK, and return is passed to the address specified in the interrupt return address (IOBIRA). In either case, it should be noted that the dispatcher always unstacks and gives control to any stacked IOBLOKs and CPEXBLOKs prior to dispatching a user. This guarantees that CP information needed by a virtual machine (such as page availability) is always as up to date as possible.

CP Spooling

The spooling support in CP performs three functions:

- Simulates the operation of the virtual unit record devices that are attached to each user's virtual machine configuration. The simulation is done in such a way that it appears to the program in the virtual machine that it is controlling a real unit record device. This support involves the interception and interpretation of virtual machine SIOs, the movement of data to and from the virtual machine's virtual storage space, and the reflection of the necessary interruption codes and ending conditions in PSWs, CSWs, and sense bytes. This support is provided by the virtual spooling executive.

- Operates the real unit record equipment, attached to the system, that transcribes virtual machine output spool files to the real printer or punch and input from the real card reader to DASD storage. This function is provided by the real spooling executive.

- Provides an interface among the virtual machines, the system operator, and the spooling system so that the location, format, priority and utilization of the systems spooling data and resources can be controlled.

Spool Data and File Format

Data Format

The buffers that collect and write spool data are all one page (4096 bytes) in length, and contain the data to be transcribed and all CCWs necessary for operating the unit record devices that perform the transcription. The data is provided in the exact format required with no compression except that trailing blanks are suppressed. A two byte field at the end of the data CCW's contains the length of the original data (including the truncated blanks.) The first two doublewords of each buffer contain linkage information described below, followed by the data and CCWs, except for the first spool buffer which contains 3800 printer-related information.

Spool files created on virtual spooled 3211, 3203, 3262, 3289E, or 4245 printers may contain the following CCWs:

```
Load FCB   (X'63')
Fold       (X'43')
Unfold     (X'23')
```

Spool files created on a virtual 3800 printer may contain the following CCW's:

```
Load Forms Control Buffer     (X'63')
Load Translate Table          (X'83')
Load WCGM                     (X'53')
Load Copy Number              (X'23')
Load FOSC                     (X'43')
Load Graphic Char. Mod.       (X'25')
Load Copy Modification        (X'35')
Initialize Printer            (X'37')
Clear Printer                 (X'87')
Select Translate Table 0      (X'47')
Select Translate Table 1      (X'57')
Select Translate Table 2      (X'67')
Select Translate Table 3      (X'77')
End of Transmission           (X'07')
Mark Form                     (X'17')
```

In addition, since the data associated with the Load Graphic Modification (X'25') and Load Copy Modification (X'35') CCW's may be longer than 4080 bytes, these data may appear in successive DASD buffers. This is accomplished by setting the `Data Chaining` bit in the CCW associated with each section of data except the last one.

Each spool logical record (card or print line) is stored as one CCW that moves data (READ or WRITE), a TIC to the following CCW, and the full data record. Space is left at the end of each buffer so that a SENSE command can be inserted to force concurrent channel end and device end. For card punch channel programs there is an additional back chain field that points to the card previously punched so that error recovery for punch equipment checks can back up one card. The only exception to the format of READ/WRITE-TIC-Data is in buffers of files directed to the printer. In this case, immediate operation code CCWs (skips and spaces) are followed by the next CCW.

File Format

In addition to the data and CCWs contained in each spool buffer, the first two doublewords contain forward and backward links to the next and previous buffers in the file. This two-way linkage allows the file to be backspaced or restarted from any point at any time. The exception to this are spool files that contain 3800 printer-related CCW's. These files can only be restarted from the beginning of the file. Also, it means that if I/O errors are encountered while reading one buffer, the file is put in system hold status. If purged, all buffers except those in error are released. The two-way chain allows this control of the file while preventing fragmentation by allowing pages to be assigned and released individually regardless of their ownership.

The first spool buffer of an output spool file contains a special data record called the tag record. This record immediately follows the two doublewords containing the forward and backward buffer linkage pointers. The tag record allows VM/SP users to specify information to be associated with spool files that they generate. The information is entered via the CP TAG command, although the tag record is not considered a spool file data record and is not printed or punched as part of the spool file. However, the contents may be interrogated via the CP TAG QUERY command. There are two fields at the end of the buffer to identify the number of buffers in the file and the size of the largest data written in the file.

The format of the tag record is a NOP CCW, followed by a TIC to the next CCW and a 136-byte data field. Any blanks contained in this NOP CCW will not be truncated. Therefore, the two byte length field at the end of the CCW will always contain X'0088' (decimal 136). To differentiate the tag record from an immediate NOP CCW (no TIC-data sequence) independently of the command code, the skip bit (bit 35) in the CCW has the following convention:

```
Bit 35 = 0   for NOP CCW, TIC, data (tag record)

       = 1   for NOP CCW (immediate NOP command)
```

Each spool file in the system is controlled by a spool file control block (SFBLOK) that is resident in storage. While the file is open, these blocks are chained from the devices (either real or virtual) that are processing the file, and from device type file anchors after the file is closed. There is one file chain each for printer, reader, and punch files. Each SFBLOK contains information about the file that describes its owner and originator (these can be different for transferred files), the filename and filetype, and the class and number of copies for output files. All of these attributes can be examined and most can be changed by the file's owner or the system operator. The SFBLOK also contains information such as the starting and ending buffer addresses for the file, the starting address of the external attributes buffer if any, the length of the external attributes buffer, the record size, certain file status flags, destination, etc.

Spool Buffer Management

Real/Virtual Storage Management

Buffers that temporarily store spool data on its way between DASD secondary storage and the user's virtual machine are allocated from a pool of virtual storage space that belongs to CP. The size of this pool varies with the real storage available to VM/SP (the storage specified at system generation or actual real storage, whichever is less). Allocation is as follows:

Storage Size Available	Virtual Buffers Allocated
384K to 655,360 bytes	128
655,361 bytes to 1.1 megabytes	320
1.1 megabytes to 3 megabytes	640
over 3 megabytes	1280

Virtual storage buffers are allocated in 1-page increments by DMKPGT at the time the spool file is opened for either input or output. If no virtual storage space is available, the virtual machine is terminated with an abend. This places limits on the number of concurrent spooling operations permitted by the system because spooling operates as a high-priority task.

Real storage is not allocated for a spooling buffer until a virtual machine actually issues a SIO that attempts to transfer data between the buffer and the user's virtual storage space. At this time, a page of real storage is allocated to the buffer via the real storage paging manager. The buffer is locked in main storage (that is, is unavailable to be paged out) only for the amount of time necessary to transfer the data. After the data transfer is complete, the buffer is treated as a normal page of virtual storage, and can be selected to be paged out. This ensures that low-usage spool files do not have buffers in real storage, while the buffers for high-usage files should remain resident. (Two spool file buffers are maintained for printing on a real 3800 printer.)

DASD Space Allocation

While a spool buffer is inactive, it resides in real storage or on the paging device. After it has been filled with data from the virtual machine or a real input reader, it is written to a page of secondary DASD storage. The allocation of pages on the spooling disk(s) is managed by DMKPGT, which handles requests for both pages of virtual storage and semipermanent spool file residence. DMKPGT maintains separate allocation block chains for virtual storage and spooling pages. Each block contains control information and a bit map that allocates pages on a single cylinder. If none of the cylinders allocated have any available pages, DMKPGT enters its cylinder allocation routine.

DMKPGT attempts to even out the spooling and paging I/O load by allocating cylinders across channels and devices. To minimize seek times on a given device, cylinders are allocated as close to the relative center of the spooling or paging area as possible.

Paging Device Support: All actual I/O for the page buffers on any device is controlled by the paging I/O executive DMKPAGIO.

Virtual Spooling Manager (DMKVSP)

The two functions of the virtual spooling manager are (1) to simulate the operation of all spooled unit-record devices attached to the user's virtual machine, and (2) to read and write the spool files associated with those devices. The following virtual devices are supported for spooling, with the exceptions noted:

- IBM 2540 Card Reader/Punch, except for punch feed read and column binary
- IBM 3203 Printer Model 4 and Model 5 (132 positions)
- IBM 3262 Printer Models 1, 5 (in 3262-1 Emulator Mode), and 11 (132 positions)
- IBM 1403 Printer Models 2 and N1 (132 positions)
- IBM 3211 Printer (150 print positions)
- IBM 3505 Card Reader (except for mark senses reading)
- IBM 3525 Punch (except for the card read, print, and data protect features)
- IBM 3800 Printing Subsystem (204 print positions)
- IBM 4245 Printer Model 1 (132 positions)
- 4248 Printer Model 1 (168 positions).

The following consoles are supported for spooling when entered into the directory as the virtual system console:

- IBM 1052 Printer-Keyboard, Model 7 (via the 2150 Console)
- IBM 3210 Console Printer-Keyboard, Models 1 and 2
- IBM 3215 Console Printer-Keyboard, Model 1.

All virtual printers, except 3800, must have the universal character set feature. No checking is done on the spooled printer data. There is, however, an exception to this. On a virtual 3800 printer all Load Checks and Data Checks are reflected to the virtual machine unless explicitly disabled via the NODATCK parameter of the DEFINE command or via a `Block Data Check` CCW, or if a `Select Translate` CCW has associated with it a specified character arrangement table in the spool command.

However, any UCS buffer commands issued by the virtual machine (load UCS buffer, block data checks, etc.) are ignored. It is up to the user and the installation to ensure that the output is directed to the proper real printer via use of the output CLASS and FORMS described below. For the 3203, 3211, 3262, 3289E, 4245, or 4248 printer, forms control buffer (FCB) commands are accepted and simulated by means of a virtual FCB maintained by the executive. For the 4248 printer, the extended FCB is also accepted. The use of the virtual FCB is the only way to simulate end-of-form conditions reflected by the detection of a channel 9 or 12 punch. The LOAD FCB command and its associated data (FCB image) are also captured in the spool file. When the file prints on a real 3202, 3211, 3262, 3289E, 4245, or 4248, according to the FILefcb, CFILefcb, or DEFfcb option on the START command, the FCB image is sent to the real printer and controls channel skipping. If the file prints on a real 1403, the FCB is ignored. Since there is no provision for FCB loading, the device is carriage tape controlled.

The FOLD and UNFOLD commands are also captured in the spool file for a virtual spooled 3203, 3211, 3262, 3289E, 4245, or 4248 and sent to the real printer when the file is processed (unless it is a real 1403 printer).

For a virtual 3800 Model 1 or Model 3 printer, all Load and Control CCWs may be included in the spool file and issued when printed on a real 3800 Printing Subsystem Model 1 or Model 3. When that file prints on another device (such as 1403, 3211, 4248, or another 3800 Printing Subsystem model,) these CCWs are changed to NO-OPS and therefore ignored.

If any of the unsupported unit record features are required, the real device must be attached directly to the user's virtual machine. Thus, a 3505 reader could be a spooling input reader, but attached directly to a batch virtual machine when it is necessary to read mark sense cards.

If a 3211 type printer is started with the DEFfcb option, all LOAD FCB commands imbedded in a spool file are NO-OPed (that is, they are not sent to the printer).

Note: If a spool file contains an extended FCB, it can only be printed on a real printer that supports the extended format (the 4248), or on a real printer that NO-OPs the LOADFCB command.

Output File Processing

DMKVSP receives control from the virtual I/O executive, DMKVSI, when the user's machine issues a SIO to a spooled unit record device. DMKVSI does not pass control until it has been determined that the device is available (that is, it is not busy and has no interruptions pending). DMKVSP first determines if the device is currently processing a file. If it is, processing continues. If this is the first command issued by the given device, a new output file must be opened. An open subroutine is called to build the control blocks necessary to manage the file and to obtain virtual storage and DASD buffer space. Control is then returned to DMKVSP.

Before the first record of an output spool file is written, DMKVSP writes a tag record (NOP CCW, TIC, data sequence) and initializes the 136-byte data area to blanks. It then sets the spool buffer displacement pointer to the first doubleword in the buffer beyond the tag record. DMSVSP then analyzes and interprets the channel program associated with the virtual machine's SIO. Each CCW is tested for validity of command, address, flags, alignment, protection, etc., and if the CCW is valid, the virtual machine's data is moved from his own virtual storage space to the buffer in the spooling virtual storage. When this buffer is full, it is written to a page of DASD secondary storage and a new buffer is obtained. The interpretation of the virtual machine's channel program continues until there are no more CCWs or until an error condition is detected that prohibits further processing. In either case, the device is marked as having the proper interruptions pending, a CSW is constructed, and DMKVSP exits to the main dispatcher. In contrast to nonspooled I/O, the virtual machine has remained in a pseudo-wait (IOWAIT) for the time it took to interpret the entire channel program.

The output file can be logically closed by the virtual machine either by issuing an invalid CCW command code, or by the CP CLOSE command. In either case, DMKSPL checks for tag record information and 3800 printer-related information in the VSPXBLOK. (The VSPXBLOK, pointed to by the VDEVEXTN field of the VDEVBLOK for the output spool device, contains the tag information entered via the CP TAG command.) If tag data exists, the first spool buffer for the file is read in, the tag data is inserted in the tag record, and the buffer is rewritten to DASD storage. If no tag data exists, the tag record data field is left blank. The first buffer is read in and the number of buffers and the size of the largest CCW is put into the first SPLZNK. The device is then cleared of pending interruptions, the file chains are completed, and the file is either queued for output on a real device of the proper type (printer or punch), or, if TRANSFER is in effect, is queued for input to another virtual machine. The 3800 printer-related information includes:

CHARS - character arrangement table
MODIFY - copy modification name
FCB - forms control buffer
FLASH - flash count overlay use

This information is contained in the VSPXBLOK for a virtual printer. When the file is closed, the information is contained in the first DASD buffer.

Input File Processing

Input file processing is similar to output file processing, except for the open and close functions, and the analysis of CCW commands and the direction of data movement. Many common routines are utilized to locate and verify CCWs, obtain buffer space, and to move the spooling data.

The difference in the open function is that instead of creating a new file, it is necessary to locate a reader file that already exists in the system. To do this, the open subroutine scans the SFBLOKs chained from the anchor, ARSPRD, to find a file with an owner userid that matches that of the caller and is not in hold status. If a file is not found, a unit check or intervention required condition is reflected to the virtual machine; otherwise, its

SFBLOK is chained to the control block for the reader and the channel program is interpreted in the same manner as for an output file.

After the input file is exhausted, a unit exception is reflected to the user machine, unless the user has requested either continuous spooling or that an EOF not be reflected. With continuous spooling, the unit exception is not reflected until the last file for that virtual machine is processed. If NOEOF is specified, the simulation terminates with a unit check or intervention-required condition (similar to what happens if the EOF button on a real reader is not pushed).

In either case, the input file is then deleted from the system, unless the user has specifically requested that his input files be saved. If the file is saved, it can be re-read any number of times.

Virtual Console Spooling

Support of virtual console I/O for both the virtual machine and VM/SP is provided as an option for the VM/SP spooling capabilities. This support fulfills the following requirements:

- Provides hardcopy support for CMS Batch Facility virtual machines
- Provides hardcopy support for display devices used as system or virtual machine consoles
- Allows disconnected virtual machines to spool virtual console output, CP commands and system resources to disk instead of losing the output
- Improves the performance of virtual machines that currently produce a large amount of console output.

Whenever a SIO is issued to a virtual machine console, the virtual console manager (DMKVCN) determines if the spooling option is active. If it is, control is passed to the virtual spooling manager to insert the data into a spool file buffer. While console spooling utilizes basically the same code as printer spooling, the following exceptions are made:

- A skip to channel 1 CCW is inserted after every 60 lines of output.
- The operator's virtual console spool buffer is written out after every 16 lines of output.
- The virtual spool buffer is written out to the allocated spool device when the first CCW is placed in that virtual buffer. The linkage area of the virtual spool buffer takes the form of a CLOSE file to allow checkpoint (DMKCKP) to recover the active spool file in the event of a shutdown because of system failure. If data in the virtual buffer has not yet been written to the spool device, it will not be recovered.

  To maintain a pseudo closed file status for console spool files, DMKSPL now assigns spool identifications to all output spool files where they are first queued.

A virtual system reset, device reset, or IPL *does not* close the virtual console spool file. The LOGOFF, FORCE, or DETACH of virtual console commands *does* close the virtual console spool file. The SHUTDOWN command does close the operator's console spool file. If the SHUTDOWN command is issued by a Class A user other than the operator, the console spool file for both the user and operator is closed.

The inclusion of the spool file tag record in a virtual console spool file is processed by DMKVSP and DMKSPL as described for printer spool files in "Output File Processing" on page 194 under "Virtual Spooling Manager (DMKVSP)" on page 193. Virtual console I/O is not spooled when the TERMINAL CONMODE 3270 function is used and when virtual channel command codes `X'19'`, `X'29'`, and `X'2A'` are used by full screen applications.

Real Spooling Manager (DMKRSP)

Command chaining is used for all unit record channel programs so that the devices are running at their maximum speed with a minimum of interruptions. In addition, because of the high speed of the 3800 Printing Subsystem, a double-buffering arrangement is used to write output to it. All other real spooling devices utilize a single output buffer.

Output File Processing

Both the input and output operations of DMKRSP are interruption driven. Thus, DMKRSP does not process unless an internally or externally generated not-ready to ready device end interruption occurs. External interruptions are generated by the hardware in the normal manner, while internal, `pseudo interruptions`, are generated by the software when an output file has been queued on the real printer or punch file chain, or when the operator issues a START command to a drained device.

Upon receipt of the initial device end for a printer or punch, DMKRSP searches the appropriate file chain for the SFBLOK of a file whose class, form, and destination matches that of the device that was made ready. If FLASH is specified for a 3800 printer, and the file has not been/is converted, the flash overlay name must also match. In addition, the location of 3800 printer Load CCWs within the spool file will determine whether it is eligible for printing on the given output device. If the device is in AUTO or SETUP mode, first preference is given to the current form. If no file with the current form exists, a file with a different form is selected, and a MOUNT REQ message is sent to the operator. On the next device end, processing will continue. When the SFBLOK is located (provided the file is not in a hold status), it is unchained from the output queue and chained to the real device block that services the file. A message is sent to the operator to indicate the file is printing. A page of real main storage (two pages for a 3800 printer) is then obtained for use as a buffer, and the output separator routine (DMKSEP) is called to print output identifier pages. DMKTCS and DMKTCT are then called to set up the 3800 printer for printing that file. When DMKSEP returns control to DMKRSP, the first buffer of the file is paged into real main storage, and the CCWs in the channel program that it contains are adjusted so that their data addresses correspond to the real addresses at which the data resides. To reduce the number of channel commands that cause movement of the printer carriage, DMKRSP also performs CCW optimization. Whenever a sequence of two carriage control commands can be replaced by a single equivalent command, the first CCW is replaced by the equivalent command and the second is changed to a no-op. For example, if a "Write and Space 1 line" command is followed by an "Immediate Space 2 lines" command, DMKRSP will replace them with a single "Write and Space 3 lines" command. The real SIO supervisor (DMKIOSQR) is then called to start the channel program, and DMKRSP exits to the dispatcher (DMKDSPCH) to await the interruption.

In SETUP mode, only a page of data at a time is printed until the START command is entered again. Then the entire file is printed.

When the channel end/device end interruption for the completed buffer is unstacked to DMKRSP, the forward chain file link field locates the next buffer. This buffer is paged-in, and the process is repeated until the final buffer is processed. At this point, the number of copies requested for the file is decremented. If the number of copies is 0, processing is terminated and the file is deleted from the system; otherwise, the process is repeated as many times as necessary. For a 3800 printer, double buffering is maintained so that the second buffer is filled while the first buffer is being printed.

When file processing is complete, a scan of the appropriate output queue is again made, and if a file is found it is processed. If the queue is empty, or if a file with a matching class, matching form, or matching destination is not found, an exit is taken to DMKDSPCH to wait for another ready interruption. If a 3800 printing device is used, the file is placed on the 3800 delayed purge queue. If this queue reaches maximum size, the oldest file in the queue is deleted from the system.

Output file processing can be modified by either the system operator, by a spooling support command or as a result of system errors. The operator commands allow a given file to be backspaced or restarted, and the files of individual users or the whole system to be held and released for output. I/O errors also affect the spooling system, and a description of how they are processed is in the section "Spool File Error Recovery" on page 204.

Input File Processing

Reader file processing is initiated by the receipt of a device end interruption from a spooling card reader. No explicit operator command is required to start the processing of an input file. When the device end is unstacked to DMKRSP, a call is made to DMKRSTIN to handle the input process. DMKRST contains an open subroutine that is called to build the necessary control blocks and to obtain the virtual, real, and DASD buffer space required for the file. A channel program to read 41 cards is built in the buffer, and DMKIOSQR is called to start the reader.

When the interruption for the first buffer is unstacked, the first card is checked for its validity as a userid card. The minimum information that this card must contain is the userid of the owner of the input file. It may appear anywhere on the card, with the restriction that it must be the first information punched. Optional information on the userid card can include a filename and type and/or the class of the virtual card reader to which the file is to be directed. If the userid is valid, the file processing continues; otherwise, the operator receives an error message and processing is terminated.

After each file buffer is read, it is written onto disk by the paging I/O routines in the same way that virtual output files are handled. When a unit exception signaling physical end of file is received from the reader, the file is closed by writing the final buffer to disk and completing and queuing the SFBLOK to the reader's file chain. If the owner of the file is currently logged on, he is given a message indicating that a file has been read and if he has an available card reader, it is posted with a device end interruption. An available reader is one of the correct class which is ready, is not busy, has no active file, and has no pending interruptions.

As DMKRST completes each phase, it returns to DMKRSP for an exit to the dispatcher (DMKDSPCH).

Accounting Card Processing

Various routines in CP accumulate, format, and store account records that contain system usage information for certain users. These routines format the information into an 80-column card image record.

In addition to the records generated by CP to account for a virtual machine's use of system resources, the user may request records in order to account for the use of virtual machine resources by jobs running under his userid. In order to do so, the user must have the account option (ACCT) entered into the directory.

The user can issue a code X'004C' DIAGNOSE instruction with a pointer to either a parameter list containing user-specified `charge to` information, or a data area containing up to 70 bytes of user-specified information to be included in the accounting record. DMKHVC validates the instruction operands, builds an account buffer (ACNTBLOK), and DMKACOQU is called to put the records in spool format.

Spooling Commands

The spooling commands provide an interface between the user, the system operator, and the spooling system. There are three types of spooling commands:

- Those that affect virtual devices
- Those that affect real devices
- Those that affect spool files that are queued within the system.

The commands that affect virtual devices are generally available to all system users, and a user can only affect the status of devices that are attached to his own virtual machine. Commands that affect the status of the real system's spooling devices can be used by the system operator only. Commands that affect closed spool files that are awaiting processing are generally available to all users, with some additional capabilities assigned to the system operator. For example, a user may alter the characteristics only of those files that have an owner's userid that matches his own, whereas the system operator may change any spool file in the system.

File States and Attributes

Each spool file in the system has a number of attributes that are assigned to it, either explicitly or by default, at the time that it is created. These attributes and their values are as follows:

- Filename and filetype can be 24-character fields. Either or both can be replaced by a user-supplied value.
- Spoolid number is a system-assigned number between 1 and 9900. It is automatically assigned when the file is created (input) or closed (output), and is unique within the system. The file's owner, the device type, and the id number are specified. Usually, the userid defaults to the identification of the user issuing the given command. Because the identification number rather than the filename and filetype is an identifier, duplicate user-assigned names do not present an identification problem.
- The number of logical records (cards or print lines) in the file is an integer between 1 and 16 million. For printer files, the record count also includes any immediate operation code space or skip CCWs.
- The originating user is the identification of the file's creator, if the file has been internally transferred from the originator's printer or punch to the new owner's card reader.
- The number of copies requested for an output file is between 1 and 255. Unless altered by the user or operator, it defaults to 1.
- The device type is used by DIAGNOSE for a file transferred to a reader to determine the virtual type of output device.
- CHARS for 3800 printer.
- FCB for 3800 printer.
- MODIFY for 3800 printer.
- FLASH for 3800 printer.

In addition to those attributes, a file that is queued for real output or virtual input always has a class associated with it. A class is a single alphameric character from A through Z or from 0 to 9. It controls both the real or virtual device on which the file will be printed, punched, or read, and the relative priority and sequence of output on the device. While each file is assigned a single class, each real spooling output device can be assigned from one to four classes. The device then processes only files that have a class attribute that corresponds to one of its own, and processes these files in the order that its own classes are specified.

For example, if a printer is assigned the classes A, D, 2, it processes any printer file with a class of A before it searches the printer output queue for a file with class D. All class D files are printed before class 2 files.

The output class for a file is assigned at the time the file is created and is the class that is associated with the virtual device that created it. While each real spooling device can have up to four classes, each virtual spooling device can have only one. When a user logs onto to the system, the class associated with a device is the one defined in his directory entry for that device. However, he can alter this class at any time by the SPOOL command. As files are created and closed by a device, they take on the device's output class.

Each file also has an operator form and a user form. These are one to eight characters each. The user form is defined by the user. There is a system default user form for each spool file type (printer, punch, or console). The operator form is determined from the user form by a table lookup. The table is defined by the SYSFORM macro at CP system generation. The SYSFORM macro also specifies the default form.

The operator may start a spool device in MANUAL mode for a certain form. The system will then process only spool files with that operator form. AUTO mode is also available. In this mode, all spool files are grouped by form, and each group is processed with the operator being prompted each time a new group is processed.

Each printer, punch, and console file has a destination (DEST is the keyword) assigned to it. The default is "OFF" unless a DESTination is set via the SPOOL, CHANGE, or CLOSE commands. The DESTination value is a one- to eight-character alphameric name your installation assigns.

Individual spool files and virtual printers and punches have one DESTination. Real printer and punch spooling devices can have as many as four DESTinations. For example, a real printer can be STARTed so that it handles files for DESTinations: "OFF", "PRT1", "PRT2", and "PRT3".

The same operator and user commands which use CLASS and FORM can be used to control DESTination.

After they are closed and are awaiting output, their class, form, and destination can be changed by a CHANGE command issued either by the file's owner or the system operator. The system operator can alter the system generated output class(es) of a real output device by the START command.

Output files transferred to a user's virtual reader can also be controlled by class. If the receiving user has several readers, the input to each can be limited to files of a certain class. In addition, the ORDER command allows sequencing of input files by class as well as spoolid number.

Output priorities can also be managed by altering the hold status of a file. Individual users can alter the hold status with the CHANGE command, while the system operator can change (hold or free) the files of specific individual users.

SPOOL and CHANGE commands can be used to modify the CHARS, FCB, MODIFY, and FLASH attributes of a file or a virtual printer.

Imbedded LOAD FCB commands also affect the printer selection. A spool file with an imbedded LOAD FCB command is only eligible for output to a printer that can handle the FCB length, or a printer that will NOOP the LOAD FCB command (like the 1403).

The operator may start a 3211-type printer in DEFFCB mode to remove the FCB restriction. DEFFCB causes DMKRSP to NOOP all LOAD FCB commands that are found in the spool file. A file that would normally be incompatible can then be printed.

Virtual Device Spooling Commands

These commands affect the status of a user's virtual spooling devices:

Command	Meaning
CLOSE	Terminates spooling operations on a specified device. It clears the device of any pending interrupt conditions, and for output files, updates the tag record, completes and queues the file for real output. Optional operands allow the user to specify a filename and filetype, and to override for the given file any standard FORM, HOLD/NOHOLD or COPY operands set into the output device by the SPOOL command.
SPOOL	Establishes the file attributes that apply to files created on, or read by, the given device. It establishes the class and form that will be in effect, whether: files are to be automatically held, input files are to be saved or purged after reading, and output files are to be directed to the real system printers and punches or are to be transferred to a user's virtual reader. The SPOOL command also specifies 3800 Model 1 or Model 3 printer attributes.

Note: The SPOOL command invokes the access control verification routine. See the *VM/SP Planning Guide and Reference* or *VM/SP CP for System Programming* for more information.

Real Device Spooling Commands

The operator can use these commands to control the activity of the real spooling devices:

Command	**Meaning**
BACKSPAC	Backspaces an active non buffered spooling device for either a specified number of pages (printers only) or to the beginning of the file (printers or punches).
DRAIN	Stops the operation of a specified output or input device after it has finished processing the file on which it is currently working. A printer must be drained prior to the issuance of the LOADBUF command. Unit record devices are normally drained prior to system shutdown.
START	Restart a device after it has been drained. Options allow the operator to specify the spooling output class, form, destination, and mode for the output device type of virtual 3800 printer files to be printed and output separator records. For a 3800 printer, the IMAGE, CHAR, FCB and PURGE options may also be specified.
FLUSH	Immediately halts the output on the specified device and either flushes that copy of the file from the system, or puts it into the system hold status for future processing.
REPEAT	Supplements the number of copies requested by the user for the file when it was created. The operator can specify a number from 1 to 255 that is added to the number specified by the user.
LOADBUF	Loads the universal character set buffer of the FCB of the specified printer with the specified image. If requested, the system verifies the loading by printing its contents on the affected printer.
SPACE	Forces the output on the specified printer to be single spaced, regardless of the skipping or spacing commands specified by the file's creator.

Spool File Management Commands: The spooling commands alter the attributes and status of closed spool files that are queued and awaiting processing. When a command applies to an individual file, the device type (RDR, PUN, PRT) and the spoolid number must be provided to identify the file. In most commands requiring a spoolid, the keyword CLASS, FORM, or DEST, followed by a valid spool class, form, or destination or the keyword ALL are acceptable substitutes for the spoolid number. This causes the command to be executed for all files of the given class, form, destination, or device type. The userid is the identification of the user issuing the command, except that the system operator must explicitly supply the identification of the user whose files he wishes to affect, or he must specify the keyword SYSTEM, which gives access to all files (valid for CHANGE, PURGE, ORDER, and TRANSFER commands also).

Command	**Meaning**
CHANGE	Changes the filename and filetype, the number of copies, the form, the destination, or the class of the specified file. The CHANGE command also specifies 3800 printer attributes. Any of the above attributes of a file can be determined via the QUERY command. In addition, this command can get a new spool id.
HOLD	Places, via the system operator, the specified file in a hold status. The file is not printed or punched until it is released by the system operator. The operator can hold any user files by device type.
FREE	Opposite of the HOLD command. Allows a file or group of files that were previously held to become available for processing. However, the user cannot reset a hold that was set by the operator with the HOLD command.
PURGE	Removes unwanted spool files from the system before they are printed or punched.
ORDER	Reorders the input files in a virtual card reader. It can order files by identification number, by class, by form, or by any combination of the three.
SPTAPE	Dumps output spool files to tape or loads output spool files from tape.
TRANSFER	Transfers a closed spool file to a different queue (reader/printer/punch), to a different user, or both simultaneously. Also optionally reclaims spool files back to the file's originator. In addition, this command can get a new spool id.

Note: The TRANSFER command invokes the access control verification routine. See the *VM/SP Planning Guide and Reference* or *VM/SP CP for System Programming* for more information.

Spool File Error Recovery

Unit Record I/O Errors

I/O errors on real spooling unit record devices are handled by a transient routine that is called by DMKIOS after it has sensed the unit check associated with the error on a spooling device. If appropriate, a restart CAW is calculated and DMKIOS is requested to retry the operation, in some cases waiting for a device end that signals that the failing device has been made ready after manual corrective measures have been taken. If, after retrying the operation, the error is unrecoverable, DMKIOS is informed that a fatal error has occurred. DMKIOS then unstacks the interruption, flagged as a fatal error, and passes control to real spooling executive. The routines that handle unstacked interruptions in real spooling execute only module operations that have been completed correctly or those that are fatal errors. If a fatal error is unstacked, the recovery mechanism depends on the operation in progress.

For fatal reader errors, processing of the current file is terminated and any portion of the file that has been read and stored on disk is purged. The owner of the file is not informed of the presence of a fractional part of the file in the system.

For fatal printer or punch errors, the SFBLOK for the partially completed file is re-queued to the appropriate output list and processing can be resumed by another available printer or punch, or can be deferred until the failing device is repaired.

In any case, the failing device is marked logically offline, and no attempt is made by the system to use it until the operator varies it back online via the VARY command.

If an invalid load module is specified for a 3800 printer (refer to DIAGNOSE code X'74'), the file involved is held or purged, and the printer queue is searched for the next file to print. In addition, the user and operator are sent a message describing the action.

DASD Errors During Spooling

DASD I/O errors for page writes are transparent to the user. A new page for the buffer is assigned, the file linkage pointers are adjusted, and the buffer is rewritten. The failing page is not de-allocated and no subsequent request for page space is granted access to the failing page. If an unrecoverable error is encountered while reading a page, processing depends on the routine that is reading the file. If the processing is being done for a virtual reader, the user is informed of the error and a unit check/intervention required condition is reflected to the reader. If the processing is being done for a real printer or punch, the failing buffer is put into the system hold status, and processing continues with the next file. In either case, the DASD page is not de-allocated and it is not available for the use of other tasks.

DASD Spool Space Exhausted

If the space allocated for paging and spooling on the system's DASD volumes is exhausted and more is requested by a virtual spooling function, the user receives a message and a unit check intervention required condition is reflected to the virtual output device that is requesting the space, the output file is automatically closed and it is available for future processing. The user can clear the unit check and periodically retry the operation which will start when space is free or completely restart later from the beginning of the job. If the task requesting the space is the real spooling reader task, the operator receives an error message and the partially complete file is purged. Any time the spooling space is exhausted, the operator is warned by a console message and alarm. However, the system attempts to continue normal operation.

Recovery from System Failure

Should the system suffer an abnormal termination, CP attempts to perform a warm start. Spool file and device data, as well as other system information is copied from real storage to warm start cylinders on DASD storage. If any virtual machines were enabled for VMSAVE, they are stored on DASD as specified in the DMKSNT module. When the system is reinitialized, the spool data and other system data is retrieved from the warm start cylinders and operation continues.

If the warm start data in real storage was damaged by the abnormal termination, the warm start procedure recognizes the situation and notifies the operator that a warm start cannot be performed. Another recovery method would be to attempt a checkpoint start.

The spool file recovery routines (DMKCKS, DMKCKT, DMKCKV) dynamically checkpoint on DASD storage; the status of all open reader files, the status of all closed output files, real spooling device data, and system hold queue information. This information is stored on checkpoint cylinders that are allocated, along with warm start cylinders, at system generation.

When a checkpoint (CKPT) start is requested, spool file and spooling device information is retrieved from the checkpoint cylinders. Spool file blocks are chained to their appropriate reader, printer or punch chains; record allocation blocks are reconstructed; spooling device status is restored; and, system hold queues are chained to the proper devices. System operation then continues.

If the checkpoint start procedure encounters I/O errors or invalid DASD data on the checkpoint cylinders, the operator is notified. The FORCE option of the checkpoint start performs all the checkpoint start functions except that, invalid or unreadable files are bypassed. While this is at best a partial recovery, the only other alternative is a cold (COLD) start, where all spool file data is lost.

Recovery Management Support (RMS)

The machine check handler (MCH) minimizes lost computing time caused by machine malfunction. MCH does this by attempting to correct the malfunction immediately, and by producing machine check records and messages to assist the service representatives in determining the cause of the problem.

The channel check handler (CCH) aids the I/O supervisor (DMKIOS) to recover from channel errors. CCH provides the device-dependent error recovery programs (ERPs) with the information needed to retry a channel operation that has failed.

This support is standard and model-independent on the external level (from the user's point of view there are no considerations, at system generation time, for model dependencies).

System Initialization for RMS

DMKCPJ calls DMKIOEFL to initialize the error recording at cold start and warm start. DMKIOEFL gives control to DMKIOG to initialize the MCH area. A store CPU ID (STIDP) instruction is performed to determine if VM/SP is running in a virtual machine environment, or running standalone on the real machine. If VM/SP is running in a virtual machine, the version code is set to X'FF' by DMKPRV. If the version code returned is X'FF', the RMS functions are not initialized beyond setting the wait bit on in the machine check new PSW (virtual). This occurs because machine check interruptions are not reflected to any virtual machine. VM/SP, running on the real machine, determines whether the virtual machine should be terminated.

If the version code is not X'FF', DMKIOG determines what channels are online by performing a Store Channel ID (STIDC) instruction and saves the channel type for each channel that is online. The maximum machine check extended logout length (MCEL) indicated by the Store CPU ID (STIDP) instruction is added to the length of the MCH record header, fixed logout length and damage assessment data field. DMKIOG then calls DMKFRE to obtain the necessary storage to be allocated for the MCH record area (MCRECORD), the CP execution block (CPEXBLOK), MCHAREA, and MCEL. The address of MCHAREA is put in the PSA (AMCHAREA). Pointers to MCRECORD and the CPEXBLOK are put in MCHAREA. DMKIOG puts the address of MCEL in control register 15. DMKIOG obtains the storage for the I/O extended logout area and initializes the logout area and the ECSW to ones. The I/O extended logout pointer is saved at location 172 and control register 15 is initialized with the address of the extended logout area. The length of the CCH record and the online channel types are saved in DMKCCH. It should be noted that the ability of a processor to produce an extended logout or I/O extended logout and the length of the logouts are both model- and channel-dependent. 3081 processors do not store a machine check fixed logout, machine check extended logout, or a region code. If VM/SP is being initialized on a Model 165 II or 168, the 2860, 2870, and 2880 standalone channel modules are loaded and locked by the paging supervisor and the pointers are saved in DMKCCH. If VM/SP is being initialized on any other model, the integrated channel support is assumed; this support is part of the channel control subroutine of DMKCCH. Before returning to DMKIOE, the VM/SP error recording cylinders are initialized. DMKIOE passes control back to DMKCPJ and control register 14 is initialized with the proper mask to record machine checks.

Overview of Machine Check Handler

A machine malfunction can originate from the processor, real storage or control storage. When any of these fails to work properly, the processor attempts to correct the malfunction.

When the malfunction is corrected, the machine check handler (MCH) is notified by a machine check interruption and the processor logs out fields of information in real storage, detailing the cause and nature of the error. The model-independent data is stored in the fixed logout area and the model-dependent data is stored in the extended logout area. The machine check handler uses these fields to analyze the error, format an error record, and write the record out on the error recording cylinder of SYSRES.

If the machine fails to recover from the malfunction through its own recovery facilities, the machine check handler is notified by a machine check interruption. An interruption code, noting that the recovery attempt was unsuccessful, is inserted in the fixed logout area. The machine check handler then analyzes the data and attempts to keep the system as fully operational as possible.

Recovery from machine malfunctions can be divided into the following categories: functional recovery, system recovery, operator-initiated restart, and system repair. These levels of error recovery are discussed in their order of acceptability, functional recovery being most acceptable and system repair being least acceptable:

Functional Recovery: Functional recovery is recovery from a machine check without adverse effect on the system or the interrupted user. This type of recovery can be made by processor retry, the ECC facility, or the machine check handler. Processor retry and ECC error correcting facilities are discussed separately in this section because they are significant in the total error recovery scheme. Functional recovery by MCH is made by correcting storage protect feature (SPF) keys and intermittent errors in real storage.

System Recovery: System recovery is attempted when functional recovery is impossible. System recovery is the continuation of system operations at the expense of the interrupted user, whose virtual machine operation is terminated. System recovery can only take place if the user in question is not critical to continued system operation. An error in a system routine that is considered to be critical to system operation precludes functional recovery and would require logout and a system dump followed by reloading the system.

Operator-Initiated Restart: When the errors may have caused a loss of supervisor or system integrity, the system is put into a disabled wait state. The operator is instructed to run the standalone error recovery program (SEREP) and then manually restart the system.

System Repair: System repair is recovery that requires the services of maintenance personnel and takes place at the discretion of the operator. Usually, the operator has tried to recover by system-supported restart one or more times with no success.

System/370 Recovery Features

The operation of the Machine Check Handler depends on certain automatic recovery actions taken by the hardware and on logout information given to it by the hardware.

Processor Retry

Processor errors are automatically retried by microprogram routines. These routines save source data before it is altered by the operation. When the error is detected, a microprogram returns the processor to the beginning of the operation, or to a point where the operation was executing correctly, and the operation is repeated. After several unsuccessful retries, the error is considered permanent.

ECC Validity Checking

ECC checks the validity of data from real and control storage, automatically correcting single-bit errors. It also detects multiple-bit errors but does not correct them. Data enters and leaves storage through a storage adapter unit. This unit checks each doubleword for correct parity in each byte. If a single-bit error is detected, it is corrected. The corrected doubleword is then sent back into real or control storage and on to the processor. When a multiple-bit error is detected, a machine check interruption occurs, and the error location is placed in the fixed logout area. MCH gains control and attempts to recover from the error.

Control Registers

Two control registers are used by MCH for loading and storing control information (see Figure 29) Control register 14 contains mask bits which specify whether certain conditions can cause machine check interruptions and mask bits which control conditions under which an extended logout can occur. Control register 15 contains the address of the extended logout area.

Word	Bits	Name of Field	Associated with
14	0	Check-stop control	Mch-Chk handling
14	1	Synchronous MCEL control	Mch-Chk handling
14	2	I/O extended logout control	Chan-Chk handling
14	4	Recovery report mask	Mch-Chk handling
14	5	Degradation report mask	Mch-Chk handling
14	6	External damage report mask	Mch-Chk handling
14	7	Warning mask	Mch-Chk handling
14	8	Asynchronous MCEL control	Mch-Chk handling
14	9	Asynchronous fixed log control	Mch-Chk handling
15	8-28	MCEL address	Mch-Chk handling

Figure 29. RMS Control Register Assignments

Machine Check Handler Subroutines

VM/SP Machine Check Handler module (DMKMCH) consists of the following functions:

- Initial analysis subroutine
- Main storage analysis subroutine
- SPF analysis subroutine
- Recovery facility mode switching
- Operator communication subroutine
- Virtual user termination subroutine
- Soft recording subroutine
- Buffer error subroutine
- Termination subroutine.

Initial Analysis Subroutine

The initial analysis subroutine of DMKMCH receives control by a machine check interruption. To minimize the possibility of losing logout information by recursive machine check interruptions, the machine check new PSW gives control to DMKMCH with the system disabled for further interruptions. There is always a danger that a machine malfunction may occur immediately after DMKMCH is entered and the system is disabled for interruption. Disabling all interruptions is only a temporary measure to give the initial analysis subroutine time to make the following emergency provisions:

- It disables for soft machine check interruptions. Soft recording is not enabled until the error is recorded.

- It saves the contents of the fixed and extended logout areas in the machine check record.

- It alters the machine check new PSW to point to the term subroutine. The term subroutine handles second machine check errors.

- It enables the machine for hard machine check interruption.

- If a virtual user was running when the interruption occurred, the running status (GPRs, FPRs, PSW, M.C. old PSW, CRs, etc.) is saved in the user's VMBLOK.

- It initially examines the machine check data for the following error types:

  MCIC = ZERO
  PSW invalid
  System damage
  Timing facilities damage

  The occurrence of any of these errors is considered uncorrectable by DMKMCH; the primary system operator is informed, the error is formatted and recorded, and the system enters a wait state, code 001 or 013.

- If none of the above errors are present, it checks for a channel inoperative error on a 303x processor. If such an error is detected, DMKACRCT will be called to attempt (a) to recover the failing channel(s) and (b) if the channel(s) is still not operational mark the channel(s) offline and attempt to continue system operation.

- If the instruction processing damage bit is on, it tests for the following types of malfunctions:

  - Multiple-Bit Error in Main Storage -- Control is given to the main storage analysis subroutine.

  - SPF Key Error -- Control is given to the SPF analysis subroutine.

  - Retry failed -- If the processor was in supervisor state, the error is considered uncorrectable and the VM/SP system is terminated. If the processor was in problem state, the virtual machine is reset or terminated and the system continues operation.

- If processor retry or ECC was successful on a soft error, control is given to the soft recording subroutine to format the record, write it out on the error recording cylinder, and update the count of soft error occurrences.

- If external damage was reported, control is given to the soft recording subroutine to format the record and write it out on the error recording cylinder.

Main Storage Analysis Subroutine

The main storage analysis subroutine is given control when the machine check interruption was caused by a multiple-bit storage error. An initial function points the machine check new PSW to an internal subroutine to indicate a solid machine check, in case a machine check interruption occurs while exercising main storage.

Damaged storage areas associated with any portion of the CP nucleus itself cannot be refreshed; multiple-bit storage errors in CP cause the VM/SP system to be terminated. An automatic restart reinitializes VM/SP.

If the damage is not in the CP nucleus, main storage is exercised to determine if the failure is solid or intermittent. Multiple-bit ECC storage errors on a 3031, 3032, or 3033 processor are always treated as solid errors. If the failure is solid, the 4K page frame is marked unavailable for use by the system. If the failure is intermittent, the page frame is marked invalid. The change bits associated with the damaged page frame are checked to determine if the page had been altered, by the virtual machine. If no alteration had occurred, VM/SP assigns a new page frame to the virtual machine and a backup copy of the page is brought into storage the next time the page is referenced. If the page had been altered VM/SP resets or terminates the virtual machine, clears its virtual storage, and sends an appropriate message to the user. Normal system operation continues for all other users.

Storage Protect Feature (SPF) Analysis Subroutine

The SPF analysis subroutine is given control when the machine check interruption was caused by an SPF error. An initial function points the machine check new PSW to an internal subroutine if a machine check interruption occurs during testing and validation. The SPF analysis routine then determines if the error was associated with a failure in virtual machine storage or in the storage associated with the control program.

An SPF error associated with VM/SP is a potentially catastrophic failure. Namely, VM/SP always runs with a PSW key of zero, which means that the SPF key in main storage is not checked for an out-of-parity condition. The SPF analysis subroutine exercises all 16 keys in the failing storage 2K page frame. If an SPF machine check occurs in exercising the 16 keys 5 times each, the error is considered solid and the operating system is terminated with a system shutdown. If an SPF machine check does not occur, the machine check is considered intermittent. The zero key is restored to the failing 2K page frame and this is transparent to the virtual machine.

If an SPF machine check occurs, which is associated with a virtual machine, the SPF analysis subroutine exercises all 16 keys in the failing storage 2K page frame. If an SPF machine check does not occur, the machine check is intermittent and the SWPTABLE for the page associated with the failing storage address is located. The storage key for the failing 2K storage page frame is retrieved from the SWPTABLE and the change and reference bits are set on in the storage key. The storage key is then stored into the affected failing storage 2K page frame. If an SPF machine check occurs in exercising the 16 keys 5 times each, then the machine check is considered solid and the following actions are taken:

- The virtual machine is selectively reset or terminated by the virtual machine termination subroutine;
- The 4K page frame associated with the failing address is removed as an available system resource.

This is accomplished by locating the CORTABLE for the defective page and altering the CORFPNT and CORPBPNT pointers to make the page unavailable to the system. The CORDISA bit in this CORTABLE is set on to identify the reason for the status of this page in a system dump.

Recovery Facility Mode Switching

The recovery facility mode switching subroutine (DMKMCIMS) allows the service representative to change the mode that processor retry and ECC recording are operating in. This subroutine receives control when a user with privilege class F issues some form of the SET command with the MODE operand. A check is initially made to determine if this is VM/SP running under VM/SP. If this is the case, the request is ignored and control is returned to the calling routine. For the format and usage of the SET command with the MODE operand, refer to the *VM/SP CP Command Reference.*

Operator Communication Subroutine

The operator communication subroutine is invoked when the integrity of the system has degraded to a point where automatic shutdown and reload of the system has been tried and was unsuccessful, or could not be attempted due to the severity of the hardware failure. A check is first made to determine if the system operator is logged on as a user. Next, a check is made to determine if the system operator is disconnected. If either of these checks is not affirmative, a message cannot be issued directly to the system operator. A LPSW is performed to place the processor in a disabled wait state with a recognizable wait state code in the processor instruction counter.

Virtual User Termination Subroutine

The virtual machine termination subroutine selectively resets or terminates a virtual user whose operation has been interrupted by an uncorrectable machine check. First, the machine is marked nondispatchable to prevent the damaged machine from running before reset or termination is performed. The machine check record is formatted and DMKIOEMC is called to record the error. Then the user is notified by a call to DMKQCNWT that a machine check has occurred and that his operation is terminated. The primary system operator is notified of the virtual user termination by a message issued by a call to DMKQCNWT. If the virtual machine is running in the virtual=real area, DMKUSO is called to log the virtual machine off the system and to return the storage previously allocated to the virtual machine and to clear any outstanding virtual machine I/O requests. The HOLD option of LOGOFF is invoked to allow a user on a dial facility to retain the connection and thus permit LOGON without re-establishing the line connection. However, if the virtual machine is running in the virtual area, and DMKCFM is then called to put the virtual machine in console function mode, the user must re-initialize the system to commence operation.

Soft Recording Subroutine

The soft recording subroutine performs two basic functions:

- Formats a machine check record and calls DMKIOEMC to record the error on the error recording cylinder.
- Maintains the threshold for processor retry and ECC errors and switches from recording to quiet mode when the threshold value is exceeded. To accomplish this, a counter is maintained by DMKMCH for successful processor retry and corrected ECC events.

Processor Retry Recording Mode: Recording mode (bit 4 of control register 14 set to one) is the initialized state, and normal operating state of VM/SP for processor retry errors. Recording mode may also be entered by use of the CP SET command. When 12 soft machine checks have occurred, the soft recording subroutine switches the processor from recording mode to quiet mode. For the purpose of model-independent implementation, this is accomplished by setting bit 4 of control register 14 to zero. Because in quiet mode no soft machine check interruptions occur, a switch from quiet mode to recording mode can be made by issuing the SET MODE {RETRY|MAIN} RECORD command. While in recording mode, corrected CPU {RETRY|MAIN} reports are formatted and recorded on the VM/SP error recording cylinders, but the primary systems operator is not informed of these occurrences.

Processor Retry Quiet Mode: Quiet mode (bit 4 of control register 14 set to 0) can be entered in one of two ways:

- When 12 soft machine checks have occurred

- When the SET MODE RETRY QUIET command is executed by a class F user.

In this mode, both processor retry and ECC reporting are disabled. The processor remains in quiet mode until the next system IPL (warm start or cold start) occurs or a SET MODE {RETRY|MAIN} RECORD command is executed by a class F user. SET MODE MAIN is treated as invalid on 3031, 3032, 3033, and 3081 processors.

ECC Recording Modes: To achieve model-independent support, RMS does not set a specific mode for ECC recording. The mode in which ECC recording is initialized depends upon the hardware design for each specific processor model. For the IBM System/370 Models 135, 135-3, 138, 145, 145-3, 148, 158, 168, 3031, 3032, 3033, and 3081, the hardware-initialized state (therefore the normal operational state for VM/SP) is quiet mode. For the IBM System/370 Models 155 II and 165 II, the hardware initialized state (the normal operational state for VM/SP) is record mode. An automatic restart incident due to a VM/SP failure does not reset the ECC recording mode in effect at the time of the failure.

The change from record to quiet mode for ECC recording can be initiated in either of the following ways:

- By issuing the SET MODE {MAIN|RETRY} QUIET command
- Automatically whenever 12 soft machine checks have occurred.

For the purpose of model-independent implementation, this occurs by setting bit 4 of control register 14 to zero.

The change from quiet to record mode for ECC recording can be accomplished by use of the SET MODE MAIN RECORD command. This recording mode option is for use by maintenance personnel only. It should be noted that processor retry is placed in recording mode if it is not in that state when the SET MODE MAIN RECORD command is issued.

While in recording mode, corrected ECC reports are formatted and recorded on the error recording cylinder, but the primary systems operator is not informed of these incidents.

Buffer Error Subroutine

On processor models equipped with a high-speed buffer (155-II, 158, 165-II, 168, 3031, 3032, 3033, 3081) or a data lookaside table (DLAT) (165-II, 168, 3031, 3032, 3033, 3081), the deletion of buffer blocks because of hardware failure is reported via a degradation report machine check interruption. MCH enables itself for degradation report machine check interruptions at system initialization by setting bit 5 of control register 14 to 1. If a machine check interruption occurs that indicates high-speed buffer or DLAT damage, MCH formats the record and calls DMKIOEMC to record it on the error recording cylinder, informs the primary systems operator of the failure, and returns control to the system to continue normal operation.

Termination Subroutine

The termination subroutine is given control if a hard machine check interruption occurs while DMKMCH is in the process of handling a machine check interruption. Note that soft error reporting is disabled for the entire time that MCH is processing an error.

An analysis is performed of the machine check interruption code of the first error to determine if it was a soft error. If it was, the first error is recorded, the system status is restored and control is restored to the point where the first error occurred. If the first error was a hard error, the operator communication subroutine is given control to issue a message directly to the system operator, and to terminate CP operation.

Overview of Channel Check Handler

The channel check handler (CCH) aids the I/O supervisor in recovering from channel errors and informs the operator or service representative of the occurrence of channel errors.

CCH receives control from the I/O supervisor when a channel data check, channel control check, or interface control check occurs. CCH produces an I/O error block (IOERBLOK) for the error recovery program and a record to be written on the error recording cylinder for the system operator or service representative. The operator or service representative may obtain a copy of the record by using the CMS CPEREP command. A message about the channel error is issued to the system operator each time a record is written on the error recording cylinder.

When the I/O supervisor program detects a channel error during routine status examination following an SIO, TIO, HIO, or an I/O interruption, it passes control to the channel check handler (DMKCCH). DMKCCH analyzes the channel logout information and constructs an IOERBLOK and, if the error is a channel control or interface control check, an ECSW is constructed and placed in the IOERBLOK. The IOERBLOK provides information for the device-dependent error recovery procedures. DMKCCH also constructs a record to be recorded on the error recording cylinder. Normally, DMKCCH returns control to the I/O supervisor after constructing an IOERBLOK and a record. However, if DMKCCH determines that system integrity has been damaged (system reset or invalid unit address, etc.), then CP operation is terminated. CP termination causes DMKCCH to issue a message directly to the system operator and place the processor in a disabled wait state with a recognizable wait code in the processor instruction counter.

Normally, when DMKCCH returns control to the I/O supervisor, the error recovery program for the device which experienced the error is scheduled. When the ERP receives control, it prepares to retry the operation if analysis of the IOERBLOK indicates that retry is possible. Depending on the device type and error condition, the ERP either effects recovery or marks the event fatal and returns control to the I/O supervisor. The I/O supervisor calls the recording routine DMKIOE to record the channel error.

The primary system operator is notified of the failure, and DMKIOE returns control to the system and normal processing continues.

If the channel check is associated with an I/O event initiated by a SIO in a virtual machine, the logout is reflected to the virtual machine in one of two ways, depending upon whether the channel check occurred at SIO time or later in an interrupt. If it occurred at SIO time, then DMKVSI (or occasionally DMKVIO) calls upon DMKCCHRF to reflect the logout. If it occurred in an I/O interrupt, the dispatcher notices the channel check as it is reflecting the I/O interrupt to the virtual machine, and so, at that time, DMKDSP calls upon DMKCCHRF to reflect the logout.

Channel Control Subroutine

Control is passed to the channel control subroutine of DMKCCH after a SIO with failing status stored, or an I/O interrupt because of a channel control check, interface control check, or channel data check.

If `logout pending` is indicated in the CSW, the CP termination flag is set. The existence of real device blocks (RCHBLOK, RCUBLOK, RDEVBLOK), for the failing device address, is determined by a call to DMKSCNRU and an indicator is set if they do exist. An indicator is also set if the IOBLOK for the failing device address exists. A call to DMKFREE obtains storage space for the channel check record and the channel control subroutine builds the record. If the indicators show that the real device blocks and the IOBLOK exist, a call to DMKFREE obtains storage space and the channel control subroutine builds the I/O error block (IOERBLOK); if these blocks do not exist, the IOERBLOK is not built. The IOERBLOK is used for two purposes:

- The device-dependent error recording program uses the IOERBLOK to attempt recovery on CP-initiated I/O events. If the I/O events that resulted in a channel check are associated with a virtual machine, the I/O fatal flag is set in the IOBLOK and the virtual machine is reset, cleared, and put into CP read status. The length and address of the channel check record is placed in the IOERBLOK and the IOERBLOK is chained off the IOBLOK.

- DMKIOECC uses the IOERBLOK to record the channel check record on the error recording cylinders.

The channel control subroutine gives control to a channel-dependent error analysis routine to build or save the extended channel status word (ECSW). When the channel control subroutine regains control, eight active addresses are saved in the channel check record.

If the CP termination flag is set, the I/O extended logout data from the channel check record is restored to main storage for use by SEREP. If the system operator is both logged on as a user and connected to the system, a message is sent to him advising him of the channel error. A LPSW is then executed to place the processor in a disabled wait state with a wait state code of 002 in the processor instruction counter.

If the CP termination flag is not set, a check is made to determine if an IOERBLOK was built by the channel control subroutine.

If an IOERBLOK was not built, a CPEXBLOK is stacked to call DMKIOECC to record the channel check record on the error recording cylinders and send the system operator a message informing him of the error. If channel termination is set, DMKACRCT is called to attempt to recover the failing channel with a CLRCH operation. If the CLRCH recovers the failing channel, or if the system can continue operation with the failing channel marked offline, DMKACR returns to DMKCCH. Otherwise a wait state X'002' is loaded.

If an IOERBLOK was built, control is returned to DMKIOS, which calls the appropriate ERP. Whether or not recovery is successful, DMKIOS eventually calls DMKIOE to record the channel check record. DMKIOE examines the status of the in CSW error in the IOERBLOK to determine if it was a channel error; if so, it finds the length and pointer to the channel check record and records the error on the error recording cylinder. If this was not a channel error, DMKIOE continues normal processing.

Individual Routines

A separate channel error analysis routine is provided for each type of channel for which DMKCCH can be used. The purpose of these routines and the channel control subroutine is to analyze the channel logout to determine the extent of damage and to create a sequence and termination code to be placed in the ECSW in the IOERBLOK. At system initialization, the correct model dependent channel recovery routine is loaded and the storage necessary to support the routine is allocated. The model-dependent error analysis subroutines and routines and their functions are as follows:

Integrated Channels (Models 135, 135-3, 138, 145, 145-3, 148, 155 II, 158, and 3031, 3032, 3033, 3081, and 43xx Processors)

Since all of these systems have integrated channels one common subroutine is used to handle all of these processor types. This subroutine:

- Indicates CP termination if the ECSW is not complete or the reset codes are invalid.
- If the channel has been reset or if the error was an I/O interface inoperative condition on a 303x processor, indicate channel termination.
- Moves the ECSW to the IOERBLOK.
- Moves the hardware stored unit address and the I/O extended logout to the channel check record.
- Sets the I/O extended logout area and ECSW area to ones.
- Returns control to the channel control subroutine.

2860 Channel (Models 165 II, 168)

The 2860 logout area is checked to determine if a complete logout exists; if not, CP termination is necessary.

A check is made in the logout area for validity of the CSW fields and bits are set in the channel check record's ECSW field to indicate bad fields.

The channel logout is then checked and sequence codes are set based on the presence of a channel control check, or an interface control check. If a channel control check is present, the codes set are determined through parity. The count determines if parity is good and sets a resultant condition code.

The logout area is examined to ensure that the unit address has valid parity and is the same address passed by DMKIOS. If so, the unit-address-valid bit in the ECSW is set. If the unit address is not valid, the unit-address-valid bit is reset to indicate the invalid condition.

The ECSW field in the channel check record is moved to the IOERBLOK, if one exists.

After completing the ECSW the 2680 routine moves the 2860 I/O extended logout into the channel check record, sets the I/O extended logout area to ones, and returns to the channel control subroutine.

2870 Channel (Models 165 II, 168)

If the channel failed to log out completely, at least part of the logout area is all ones. If a fullword of ones is found, a CP termination condition exists.

A check is made in the logout area for valid CSW fields, and bits are set in the channel check record's ECSW field to indicate bad fields.

The termination and sequence codes are set depending on the presence of an interface control check or channel control check. If a channel control check is present, the codes set are determined through parity, count, and/or data transfer checks. For the 2870, parity can be determined directly from the channel logout.

The logout area is also examined to ensure valid parity in the unit address and to ensure that the address is the same as that passed to DMKCCH by DMKIOS. If so, the unit-address-valid bit in the ECSW is set.

The third word of the logout area is also analyzed for type II errors. If a type II error is found, a CP termination condition exists.

The ECSW field in the channel check record is moved to the IOERBLOK, if one exists.

Before returning to the channel control subroutine, the 2870 routine moves the 2870 I/O extended logout into the channel check record and sets the I/O extended logout area to ones.

2880 Channel (Models 165 II and 168)

This routine analyzes nine words of the 28-word logout.

The 2880 analysis routine handles channel data checks, interface control checks, and channel control checks.

Termination code 3 (system reset) is not set in the ECSW because the 2880 channel does not issue system reset to the devices. Retry codes of 0 to 5 are possible.

Note: There are several catastrophic conditions under which the CP termination flag can be set, in the 2880 analysis routine. They are:

- The channel did not complete the logout.
- The CSW is not reliable.
- The unit address in the I/O interruption device address field is not correct.

Only a channel check record is needed if the channel has recognized an internal error and has recovered from it without any damage. No recovery action is necessary in these cases.

If the channel address in the I/O interruption device address field does not match the channel address in the logout, a CP termination condition exists.

If the channel was doing a scan and the unit control word had a parity check, a CP termination condition exists. If there was no parity check, there was no damage during the scan and only a channel check record is required.

Depending on the sequence the channel has entered, the termination and sequence codes are set; command address, unit address, and unit status validity is determined; and the sequence code is set valid. The ECSW field in the channel check record is moved into the IOERBLOK, if one exists.

Before returning to the channel control subroutine, the 2880 routine moves the I/O extended logout into the channel check record and sets the I/O extended logout area to ones.

Error Recording Interface for Virtual Machines

The error recording interface provides a means of recording errors encountered by operating systems running in a virtual machine under VM/SP. An SVC 76 issued by a virtual machine is used to signal CP that error recording is required. The SVC interruption handler in DMKSVD examines general registers 0 and 1 to determine if valid parameters have been passed. If valid parameters are not found, the SVC is reflected back to the virtual machine and no recording takes place. If valid parameters are passed, a pageable routine (DMKVER) processes the error record.

DMKVER validates the record passed by the virtual machine. If invalid conditions are found, no recording takes place. Control is returned to the SVC interruption routine in DMKPSA to reflect the SVC to the virtual machine as an SVC interruption. The action taken by the virtual machine is dependent on the operating system running in the virtual machine, not VM/SP. If the record is valid, it is modified by changing virtual information to real. The actual recording is accomplished by using the existing modules DMKIOE and DMKIOF.

Control is then returned to the instruction following the SVC 76 rather than reflecting the SVC. This eliminates the duplication of error recording in VM/SP and the operating system in the virtual machine. If DMKVER determines that the recording represented a permanent I/O error, a message is sent to the primary system operator.

Error Recording and Recovery

The error recording facility is made up of six modules. One module (DMKIOE) is resident and the other five (DMKIOC, DMKIOF, DMKIOG, DMKIOH, and DMKIOJ) are pageable.

The error recording modules record temporary errors (statistical data recording) for CP generated I/O except for DASDs with a buffered log.

The error recording routines record: unit checks, statistical data counter overflow records, selected temporary DASD errors, machine checks, channel checks, and hardware environmental counter sense data on the error recording cylinders of the system resident device in a format suitable for subsequent processing by the CPEREP command (DMSIFC). The recorder asynchronously updates the statistical data counters for supported devices. The recorder also initializes the error recording cylinders at IPL if they are in an unrecognizable format.

When the recorder is entered from DMKIOS, it is entered at DMKIOERR. This entry is used for unit checks and channel data checks. A test is made of the failing CSW (located in the IOERBLOK) to see if the error was a channel error. If it was, control is passed to the routine for recording channel checks.

The IOERBLOK sense data, IOBLOK flags, and VMBLOK privilege class are examined to determine if the error should be recorded.

Error Record Writing

After an error record is formatted, it is added to the error recording area using DMKRPAGT and DMKRPAPT. The error recording area has page-sized records (4096 bytes). Each page contains a header (8 bytes) which signifies: the cylinder and page number of the page on a count-key-data DASD, or page number of the page on a fixed-block DASD (4 bytes), the next available space for recording within page (2 bytes), a page-in-use indicator (1 byte), and a flag byte. Each record within the page is recorded with a 4-byte prefix.

If an error record is too large to be added into a page, a new page is retrieved, updated with record, and placed back on the error recording cylinder with the paging routines.

The area to be used for error recording is specified by the installation or system programmer at system generation time. For count-key-data DASDs, an integral number of cylinders (from two to nine) is specified. For fixed-block DASDs any number of pages can be specified. Generally, fifty pages or more is an appropriate amount. Errors are recorded in the order in which they occur. If the error recording area becomes 90 percent full, a message is issued to the operator using DMKQCNWT to warn him of the condition. If the area becomes full, another message is issued to inform the operator and recording is stopped.

On the 3031, 3032, and 3033 processors, frame records are read from all accessible SRF devices and written on the error recording area during initialization if no records exist (as after a CPEREP CLEARF operation).

If a channel check error is to be recorded, the recorder is entered at DMKIOERR or DMKIOECC. The channel check handler determines the entry. A channel check error record is formatted.

A machine check enters at DMKIOEMC. Pointers are passed from the machine check handler in registers 6 and 7 to locate a buffer where the machine check record and length are saved. A machine check error record is recorded with the saved machine check logout and additional information. The machine check error record is written onto the error recording area by using the paging routines.

Hardware environmental counter records are formed using routine DMKIOEEV. This routine is scheduled by DMKIOS after control is returned from the ERP. Sense data information is stored in the IOERBLOK by the ERP. The record formed is called a nonstandard record.

Clear and Format Recording Area

DMKIOEFM is called by DMSIFC (CPEREP command) via a DIAGNOSE instruction. DMKIOEFM is invoked to reset the specified error recording cylinders (if CLEAR, CLEARF, or ZERO=Y was specified). The clear is performed by resetting each page-header, space-available field. Pointers in storage are then updated to address the first available page in the error recording area. Control is then returned to the calling routine. For details on the CPEREP command and EREP execution, refer to the *VM/SP OLTSEP and Error Recording Guide* and OS/VS EREP publications.

CLEARF on a 3031, 3032, or 3033 processor clears the area, then causes the frame records to be read from each SRF device specified at system generation on the RIOGEN macro.

Find First Recording Cylinder at IPL

DMKIOEFL is called by DMKCPI to find the first available page that can be used for error recording. The paging routines, DMKRPAPT and DMKRPAGT, are used to read the error recording pages (4096-byte records). As each page record is read, it is examined to see if this record is the last recorded. If so, a pointer in storage is saved so recording can continue on that page record. Control is then returned to the caller. If any error recording page is in an unrecognizable format, the error recording area is automatically reformatted by CP.

DASD Error Recovery, ERP (DMKDAS or DMKDAD for Count-Key-Data) or (DMKDAU for FB-512)

Error recovery is attempted for CP-initiated I/O operations to its supported devices and for user-initiated operations to CP-supported devices that use a DIAGNOSE interface. The primary control blocks used for error recovery are the RDEVBLOK, the IOBLOK and the IOERBLOK. In addition, auxiliary storage is sometimes used for recovery channel programs and sense buffers.

The initial error is first detected by the I/O interruption handler which performs a SENSE operation if a unit check occurs. Unit check errors are then passed to an appropriate ERP. If a channel check is encountered, the channel check interruption handler determines whether or not retry is possible and passes control to an ERP through the I/O interruption handler. DASD errors are processed as described below.

Channel Errors

- I/O interface inoperative on a 3031, 3032, or 3033 processor is reflected to the virtual machine if the channel is dedicated. Otherwise, a call is made to DMKACR to attempt to recover the failing channel. If recovery from the channel check is not possible, a wait state X'002' occurs.
- Channel control check is treated as seek check. It is retried 10 times.
- Interface control check is treated as seek check. It is retried 10 times.
- Channel data check is treated as data check. It is retried 10 times.

Unit Check Errors

Equipment Check: Retry the operation 10 times for 3330, 3340, 3375, 3380, 3350, 2305, and FB-512 devices; twice for the 2314 and 2319. If Alternate Interface Disabled is also on, retry the operation one time. For FB-512 devices, if the error is also permanent, the command is not retried.

No Record Found (Count-Key-Data Devices Only): Execute a READ HOME ADDRESS and check home address against seek address. If they are the same, consider the error permanent. If they are not equal recalibrate and retry the channel program 10 times (2314/2319). For other devices, return to caller.

Seek Check (Count-Key-Data Devices Only): Retry the operation 10 times except that 3375/3380/3330/3350 seek checks are retried by hardware.

Intervention Required: Issue a message to console and wait for solicited device end. This procedure is repeated once.

Bus Out Check: One retry of the operation.

Data Checks: For 2314/2319 retry the operation 256 times, with a recalibrate being executed every 16th time. For the 2305/3340, retry the operation 10 times. For the 3375/3380/3330/3350/FB-512, the operation is retried by hardware.

Overrun: Retry the operation 10 times. For FB-512 devices, the operation is retried 10 times, unless the error is also permanent.

Missing Address Marker: Retry the operation 10 times.

Command Reject: The command is not retried.

File Protect: The command is not retried.

Chaining Check: Test for command reject. If not present, retry the operation 10 times.

Environmental Data Present: Issue a BUFFER UNLOAD command and retry the operation.

Track Condition Check (Count-Key-Data Devices Only): On CP I/O and DIAGNOSE I/O, when a track condition check is received from a device for which CP does not provide alternate track software recovery, the condition is treated as a permanent error. CP does provide alternate track support for other devices; this support is described in the section "Alternate Track Recovery, ERP (DMKTRK)" on page 225.

Check Data (FB-512 Devices Only): The command is not retried.

The error recovery routine keeps track of the number of retries in the IOBRCNT field of the IOBLOK. This count determines if a retry limit has been exceeded for a particular error. On initial entry from DMKIOS for an error condition, the count is zero. Each time a retry is attempted, the count is increased by one.

The ERP preserves the original error CSW and sense information by placing a pointer to the original IOERBLOK in the RDEVBLOK. Additional IOERBLOKs, which are received from DMKIOS on failing restart attempts, are discarded. The original IOERBLOK is thus preserved for recording purposes.

If after a specified number of retries, DMKDAS or DMKDAD (for count-key-data devices) or DMKDAU (for FB-512 devices) fails to correct the error, the operator may or may not be notified of the error. Control is returned to DMKIOS. DMKIOS is notified of the permanent error by posting the IOBLOK (IOBSTAT=IOBFATAL). The error is recorded via DMKIOS by DMKIOERR, if DMKDAS or DMKDAU and DMKIOE determine that the error warrants recording.

If the error is corrected by a restart, the temporary or transient error is not recorded. Control is returned to DMKIOS with the error flag off.

Before returning control to DMKIOS on either a permanent error or a successful recovery, the ERP frees all auxiliary storage gotten for recovery CCWs, buffers, and IOERBLOKs, and updates the statistical counters for 2314 and 2319 devices.

The DMKIOS interface with the ERP uses the IOBSTAT and IOBFLAG fields of the IOBLOK to determine the action required when the ERP returns to DMKIOS.

When retry is to be attempted, the ERP turns on the restart bit of the IOBFLAG field. The ERP bit of the IOBFLAG field is also turned on to indicate to DMKIOS that the ERP wants control back when the task has finished. This enables the ERP to receive control even if the retry was successful and allows the freeing of all storage gotten for CCWs and temporary buffers. The IOBRCAW is set to the recovery CCW string address.

In handling an intervention-required situation, the ERP sends a message to the operator and then waits for the device end to arrive. This is accomplished by a return to DMKIOS with the ERP bit in the IOBFLAG field set on and the IOBSTRT bit in the IOBFLAG field set off. When the device end interruption arrives, the original channel program which was interrupted is then started.

The ERP flags of the IOERBLOK are also used to indicate when special recovery is being attempted. For example, a READ HOME ADDRESS command when a no record found error occurs.

The other two indicators are explained in Figure 30.

IOBFLAG IOBERP	IOBFLAG IOBRSTRT	IOBSTAT IOBFATAL	Action To Be Performed by DMKIOS
1	0	0	Return control when solicited device end arrives
1	1	0	Restart using IOBRCAW
0	0	1	Permanent I/O error
0	0	0	Retry successful

Figure 30. Summary of IOB Indicators

If the error is uncorrectable or intervention is required, the ERP calls DMKMSW to notify operator. The specific message is identified in the MSGPARM field of the IOERBLOK.

Alternate Track Recovery, ERP (DMKTRK)

The software alternate track recovery support described in the following paragraphs applies only to the 3340/3344 disk. For 3380, 3330, and 3350 disks no software support is needed since the hardware performs alternate track recovery. No support is needed for the 2305 drum since the CE is able to rewire the device to use spare tracks in place of defective tracks. For the 2314 and 2319 disks no true alternate track recovery is provided by CP. But track condition checks from any device type are reflected back to the virtual machine. Therefore, even though CP itself cannot use a 2314 or 2319 cylinder that contains a defective track, it is possible for a virtual machine to use such a cylinder if it provides its own error recovery. To facilitate this, the VM/SP version of the IBCDASDI (or Device Support Facilities-Stand-alone,5747-DS1) program allows 2314 and 2319 minidisks to be formatted with an alternate track cylinder as the last cylinder of each minidisk rather than using the last cylinders of the real disk for this purpose.

Overview of 3340 Alternate Track Support

The 3340 alternate track support applies to CP I/O, to DIAGNOSE I/O (thereby giving alternate track support to CMS), and to SIO executed in a virtual machine. For CP I/O and DIAGNOSE I/O, the alternate track recovery support essentially consists of directing (seeking) an interrupted channel program to an alternate track and restarting it. Later, in some cases, the interrupted channel program is directed back to the original cylinder and restarted there. For SIO in a virtual machine, the operating system in the virtual machine provides its own error recovery when CP reflects a track condition check to the virtual machine.

On the 3340 disk, alternate tracks are assigned in the conventional alternate tracks cylinders at the high end of the real disk, not in the last cylinder of each minidisk. Therefore a virtual machine may need to seek outside of its minidisk extent. This occurs when an operating system in a virtual machine performs its own error recovery following a track condition check. So for SIO issued from a virtual machine, CP's alternate track support must permit the virtual machine to escape from the confines of its minidisk to get to the alternate tracks assigned to the defective tracks of that minidisk. Yet at the same time CP must still prevent the virtual machine from accessing other tracks that it does not own.

Since alternate tracks are assigned only in the conventional alternate tracks cylinders at the high end of the real disk, CP does not apply minidisk cylinder relocation values to a virtual machine's channel commands that reference alternate tracks. Similarly, CP does not unrelocate alternate track CCHH addresses returned by read home address, by read record zero, in sense information, or for error recording.

Alternate Track Hardware Operation and Implications

The home address record (HA) on any track contains a flag byte with two bits that are involved in alternate track assignments. One bit, when set to one, indicates that the track is defective and that the track should have (and ordinarily does have) an alternate track assigned. The count field of record zero of a track with this bit set should point to (have the CCHH address of) the assigned alternate track. The second bit in the flag byte, when set to one, indicates that the track in which it appears is an assigned alternate track. The count field of record zero of an assigned alternate track should point back to (have the CCHH address of) the flagged defective track that it is assigned to.

Before using the pointer in record zero of a flagged track to get to the corresponding alternate, it is considered good form for an operating system to check the pointers both ways to see that each points to the other. CP performs two-way checks of the pointers for seeks to an alternate track initiated by DIAGNOSE or by SIO in a virtual machine. For its own I/O, CP uses the forward record zero pointer without performing a two-way check. Performing a two-way check would decrease performance and should not be necessary since all of the record zero pointers were checked in both directions by the Format/Allocate program (DMKFMT) when the CP-owned disk was originally formatted.

Note: The DASD Dump/Restore (DDR) program also checks the record zero pointers both ways when a tape is restored to a disk.

Except for those channel commands that deal specifically with the home address and record zero, any attempt to search or read or write on a track that is flagged as defective results in a unit check with `track condition check` indicated in the sense data.

Operations on an assigned alternate track can also result in a unit check with `track condition check` indicated in the sense data. But in this case it occurs when an attempt is made to *leave* the assigned alternate track, not when the operation is reading or writing on the track. The situations where trying to leave the alternate track results in a track condition check are:

- Any multi-track operation
- A record overflow operation.

The hardware does *not* generate a track condition check when a seek is used to leave the track. This applies to any kind of seek, including seek head.

When a channel program from a virtual machine SIO (or from a DIAGNOSE) is allowed to access an alternate track, subsequent CCWs in the channel program must be prevented from accessing adjacent tracks in the alternate track cylinder since these may belong to other virtual machines. A channel program may attempt a transition from one track to the next by any of the following:

- Seek
- Seek head
- Multi-track search or read
- Record overflow.

The full seek causes no problem: since it specifies the cylinder as well as the track, it causes the channel program to leave the alternate track and to return to a cylinder within the minidisk extent. It is certain to go back to the minidisk because the seek address was verified when the virtual CCWs were translated to real.

The seek head is dealt with as follows. When a seek to an alternate track is encountered in a virtual channel program by CP during the CCW translation process, CP converts all seek head commands (in the real, translated CCWs) to an invalid CCW opcode (X'FF'). Then when the translated channel program is executed, it is interrupted (with a command reject) at each seek head CCW so that the track to which the channel program is seeking can be checked to see that it really belongs to the virtual machine that requested the I/O. Note that this only happens to channel programs that seek out of the minidisk to an alternate track.

The multi-track operations and record overflow operations also cause no problem, because, as explained above, these are caught by the hardware and result in a track condition check.

If record overflow processing is involved and sense indicates operation incomplete, this condition will result in a fatal error because VM does not provide alternate track recovery for overflow records on a 3340/3344.

Module Function and Control Flow

DMKTRKVA - When DMKCCWTR finds a virtual machine seeking out of its minidisk extent to what should be an assigned alternate track, it has to do a check of the backward record zero pointer to verify that the alternate belongs to that minidisk. So DMKCCWTR calls DMKTRKVA, passing the CCHH address of the alternate as input, and DMKTRKVA performs CP I/O to read record zero of the alternate and then returns the pointer found in record zero to DMKCCWTR.

DMKTRKFP - This is called by both DMKUNT and DMKVIO. Its function is to handle command rejects in channel programs initiated by virtual machine SIO when the channel program was found (by DMKCCWTR) to be seeking to an alternate track outside the minidisk extent. The command rejects result because, for these channel programs, any seek head commands have been invalidated (opcode changed to X'FF') in order to trap seek heads that might switch to another minidisk's track in the alternate track cylinder.

Note: Even though DMKCCWTR may also find DIAGNOSE I/O channel programs that seek directly to an alternate track and invalidate the seek head opcodes on these channel programs, the command rejects resulting from these channel programs are handled by DMKTRKIN, not by DMKTRKFP.

DMKTRKIN - This routine performs alternate track recovery for CP I/O and for DIAGNOSE I/O both when the DIAGNOSE channel program results in a track condition check and when a command reject results from a seek head whose opcode DMKCCWTR made invalid. The routine has nothing to do with alternate track recovery for SIO issued by a virtual machine. But it does share a few small subroutines with DMKTRKFP.

DMKTRKIN is called only by DMKDASER, which in turn is called only by DMKIOS. These three routines work closely together during alternate track error recovery and the control flow back and forth between these routines is controlled to a great degree by flags in the IOBLOK and the IOERBLOK.

The control blocks of major concern in this area are the RDEVBLOK, the IOBLOK, and the IOERBLOK. When an error occurs and DMKIOS makes the initial call to DMKDASER (at the time of the first error associated with this IOBLOK), an IOERBLOK containing sense data has already been created; the IOBIOER field of the IOBLOK points to it. When DMKDASER gets control, it notices that this is a first call and it moves the pointer out of IOBIOER into RDEVIOER so that this first IOERBLOK, associated with the original error, can be kept over a period of time during which attempts may be made to retry the I/O operation. During these retries, further errors may cause new IOERBLOKs, pointed to by IOBIOER, to be sent back from DMKIOS. Generally speaking, RDEVIOER continues to point to the original IOERBLOK and new IOERBLOKs are created and sent back from DMKIOS after each retry that ends with an error. Generally, the new IOERBLOK from the failed retry is discarded before the next retry. But occasionally a new IOERBLOK is used by DMKDASER or DMKTRKIN to replace the original IOERBLOK, so it is pointed to by RDEVIOER and the first original IOERBLOK is discarded before the next retry. This happens when the new error is deemed to be more severe than the original (DMKDASER gives priority to channel checks) or when the original error gets corrected by a retry, but then the channel program fails on a later CCW (DMKTRKIN does this).

Control flow back and forth between DMKIOS and DMKDASER is controlled by the setting of the flags IOBERP, IOBRSTRT, and IOBFATAL, and has been described earlier in the section "DASD Error Recovery, ERP (DMKDAS or DMKDAD for Count-Key-Data) or (DMKDAU for FB-512)" on page 222.

The control flow back and forth between DMKDASER and DMKTRKIN is controlled by the flags IOERRDR0 and IOERALTR and by a return code that DMKTRKIN passes back in register 1. Whenever either of the two flags is set, they cause DMKDASER to call DMKTRKIN whenever DMKDASER gets control (which in this case happens after a retry), even though there is no track condition check indicated in the new IOERBLOK. The IOERRDR0 flag indicates to DMKTRKIN that the retry being returned from was used to execute a channel program to read record zero. The IOERALTR flag indicates to DMKTRKIN that the retry being returned from is a restart of a user channel program (not strictly error recovery CCWs) that had a track condition check earlier. This means that invalidated seek head opcodes can be expected.

Details of Alternate Track Recovery for CP I/O and DIAGNOSE I/O

Once a CP I/O or DIAGNOSE I/O channel program has to be restarted because of a track condition check, the error recovery procedure invalidates (for DIAGNOSE I/O only) all seek head opcodes in the channel program and sets the IOERALTR flag (indicating that alternate track error recovery is in progress) before proceeding. The IOERALTR flag remains set whenever any portion of the users channel program is being retried, until the channel program either ends successfully or ends with a permanent error.

Note: The flag does not remain set continuously; there are breaks while the error recovery procedure takes time out to use its own channel program to read record zero (the channel program is passed back to IOS as a `retry`). At these times the IOERRDR0 flag is set instead of the IOERALTR flag.

During the further execution of a DIAGNOSE Channel program, invalidated seek head opcodes may be encountered once the IOERALTR flag is turned on. CP channel programs do not use seek head. The number of these opcodes encountered may be several, or none at all, depending on the user's channel program. Also, these invalidated seek heads may be trying to seek off of an assigned alternate track (usually to the next logical track) or they may have no involvement with flagged tracks at all, again depending on the nature of the user's channel program. Whenever the channel program is stopped by an invalidated seek head, a determination is made of whether or not it is trying to get off of an alternate track. This determination is made by looking at the current cylinder number (available in sense data taken at the time of the command reject) and seeing whether or not it falls within the alternate track cylinder area at the high end of the disk. If the seek head *was not* trying to get off of an alternate track, there is no problem and the subject channel program is restarted with a seek to the current cylinder and to the track specified by the invalidated seek head. If the seek head *was* trying to get off of an alternate track, record zero of the alternate track is read first to get the cylinder number of the defective track. Then the subject channel program is restarted with a seek to the cylinder of the defective track, but to the track specified by the invalidated seek head.

Tape Error Recovery, ERP (DMKTAP and DMKTPE)

Error recovery is attempted for user-initiated tape I/O operations to CP-supported devices that use the DIAGNOSE interface. The primary control blocks used for error recovery are the RDEVBLOK, the IOBLOK, and the IOERBLOK. In addition, auxiliary storage is used for recovery channel programs (repositioning and erase).

The interruption handler, DMKIOS, performs a SENSE operation when a unit check occurs. Tape errors are then passed to DMKTAP. If the tape device is a 3480, tape errors are passed to DMKTPE instead. The sense information associated with a unit check is contained in the IOERBLOK. If a channel check is encountered, the channel check interruption handler determines if retry is possible and passes control to the ERP through the I/O interruption handler.

When an error is encountered and ERP receives control, the tape error recovery module determines if this is the first entry into the ERP for this task. The IOBRCNT (IOB error count) field of the IOB is zero on initial entry. On this first entry, the pointer to the IOERBLOK is placed in the RDEVIOER field of the RDEVBLOK. This preserves the original error CSW and sense information for recording. Thereafter, IOERBLOKS are discarded before a retry is attempted or a permanent error is passed to IOS.

The ERP looks for two other specific conditions. If the error count field is not zero, entry must be due to a recovery attempt. Thus, it may be a solicited device end to correct an intervention-required condition or a retry attempt for either tape repositioning or channel program re-execution.

The ERP keeps track of the number of retries in the IOBRCNT field of the IOBLOK to determine if a retry limit has been exceeded for a particular error. If the specified number of retries fails to correct the error, the error is recorded and DMKIOS is notified of the permanent error by turning on a status flag in the IOBLOK (IOBSTAT=IOBFATAL).

If the error is corrected, the temporary error is not recorded and control is returned to DMKIOS with error flags all off. (Some temporary errors handled by DMKTPE are recorded.) When repositioning is required in order to attempt recovery, additional ERP flags are contained in the IOERBLOK to indicate paths for specific errors (that is, data check on write must reposition, erase, and then reissue original channel program).

All error recovery is started the same except for intervention-required errors. The IOBFLAG is turned on to indicate RESTART

(IOBFLAG=IOBRSTRT), and the IOBRCAW (IOBLOK Restart CAW) is filled with the restart channel address word. In addition, an IOBFLAG flag is turned on to indicate that the ERP is in control so that control can be returned to ERP during all tape error recovery (IOBFLAG=IOBERP). In the case of an intervention required error, the ERP sends a message to the operator, and then returns to DMKIOS with indications that tell DMKIOS the ERP is waiting for a device end on this device. This is done by clearing the restart flag and returning to DMKIOS with only the IOBERP flag on.

When ERP has determined a permanent error situation or successfully recovered from an error, all auxiliary storage obtained for recovery CCWs, buffers, and IOERBLOKs is freed before a return is made to DMKIOS (see Figure 30 on page 225 for a summary of the IOB indicators); also, the statistical counters for 2400, 3410, and 3420 devices are updated.

If the error is uncorrectable or operator intervention is necessary, ERP calls the message writer to write the specific message.

3270 Remote Support Error Recovery

Recovery from errors associated with binary synchronous lines, and the related channel and transmission control unit hardware is processed by DMKBSC. Recovery from errors associated with data and control processing by the remote station (the device) as defined by remote status and sense byte definition (see *3270 Information Display Component Description*) is processed by DMKRGA. Control blocks associated with these errors are the CONTASK, the RDEVBLOK, the BSCBLOK, the NICBLOK, the IOBLOK, and the IOERBLOK.

The interruption handler, DMKIOS, performs a SENSE operation upon detection of a unit check condition (IOERBLOK). The related sense data is analyzed as it relates to the previous operation (CONTASK or BSCBLOK, whichever is applicable). If a channel check is encountered by the channel check interruption handler, the channel check interruption (DMKBSC) procedures determine if recovery can be attempted. If it cannot be retried, that operation is aborted and an appropriate message is sent to the system operator.

DMKRGB sends a Write Structured Field command with a Read Partition (QUERY) data stream to all 3278 and 3279 display stations during ENABLE processing. The purpose of this operation is to test for the presence of color, extended highlighting and programmed symbol set features on the enabled displays. A returned unit check with command reject is indicative of a device(s) without these features. For other errors, ERP receives control and either DMKBSC or DMKRGA determines if this is the first entry into the ERP for this task. The IOBRCNT (IOB error count) field of the IOB is zero on initial entry. On this first entry, the pointer to the IOERBLOK is placed in the RDEVIOER field of the RDEVBLOK. This preserves the original error CSW and sense information for recording. Thereafter, IOERBLOKs are discarded before a retry is attempted or a permanent error is passed to IOS.

The ERP looks for two other specific conditions. If the error count field is not zero, entry must be due to a recovery attempt. Thus, it may be a solicited device end to correct an intervention-required condition or a retry of channel program execution.

The ERP keeps track of the number of retries in the IOBRCNT field of the IOBLOK to determine if a retry limit has been exceeded for a particular error. If the specified number of retries fails to correct the error, the error is recorded and DMKIOS is notified of the permanent error by turning on a status flag in the IOBLOK (IOBSTAT=IOBFATAL).

If an apparent fatal error occurs during a Write Text operation, DMKRGA makes an attempt to recover by reselecting the remote 3270, using the standard SELECT channel commands. If the SELECT CCW's succeed, DMKRGA restarts the Write Text operation. If the SELECT channel command fails, DMKRGA treats it as a permanent TP failure.

If the error is corrected, the temporary error is not recorded and control is returned to DMKIOS with all error flags off.

When ERP has determined a permanent error situation or successfully recovered from an error, all auxiliary storage obtained for recovery CCWs, buffers, and IOERBLOKs is freed before a return is made to DMKIOS (see Figure 30 on page 225 for a summary of the IOB indicators). Also, the statistical counters for 3270 are updated.

The Attached Processor and Multiprocessor Environments

Attached processor support is requested by specifying AP=YES on the SYSCOR macro. Multiprocessor support is requested by specifying MP=YES. For a complete description of system generation considerations, see the *VM/SP Planning Guide and Reference.*

CP Initialization for the Attached Processor or Multiprocessor

System/370 Principles of Operation has a detailed discussion of prefixing that is necessary for understanding the initialization done for the attached processor and multiprocessor systems.

Processor Addresses

The CP initialization routine, DMKCPI, begins normal processing by storing the physical address of the initialized processor in the PSA at location absolute zero (field IPUADDR). (Prefixing has not yet been established.) The logical processor address is computed by doing a logical OR of the physical address with X'40' and is stored in the PSA in LPUADDR. The logical value is used by the CP LOCK manager to avoid using a zero value. The physical value is used for signaling between the two processors.

If AP=YES or MP=YES was coded on the SYSCOR macro, DMKCPI uses the SIGP function to see if another processor or noninitialized processor is available. If so, its physical and logical addresses are stored in the PSA in IPUADDRX and LPUADDRX, respectively. If another processor or noninitialized processor is not available, APUNONLN is set to 1. If the multiprocessing option is installed, a message is sent to the operator.

PSA Setup

The top two 4K pages of storage are marked (in the CORTABLE) as being CP-owned and are used as the PSAs for the two processors. The addresses of these two pages are stored at PREFIXA and PREFIXB in the PSA at location absolute zero. DMKAPI copies the information from the PSA at location absolute zero to the new PSA locations. In the PSA designated for the attached processor, PREFIXA and PREFIXB are switched. Thus, on either processor PREFIXA always represents the current processor and PREFIXB the other processor. The values of IPUADDR, LPUADDR, IPUADDRX, and LPUADDRX are also switched so that IPUADDR and LPUADDR always contain the processor addresses of the current processor and IPUADDRX and LPUADDRX contain the other processor addresses.

Locking

To provide system integrity, VM/SP attached processor and multiprocessor support is designed around one global system 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. 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, nonerror paths. These functions include virtual page fault processing, the simulation of virtual I/O requests and other privileged operations, and the processing of a real I/O interrupt.

If the global system lock is needed and cannot be obtained, the function must be deferred until the global system lock is available. Deferral of the function is accomplished 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 will then use the unlock 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, a processor will spin on the global system lock until it becomes available.

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.

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.

Real Storage Management Lock (RM Lock)

The real storage management lock (called the RM lock) is a spin lock that controls access to certain real storage management functions and queues.

VMBLOK Lock

Each VMBLOK contains one lock, called VMLOCK, which is used by routines that need to serialize certain virtual machine related resources. These resources include the following:

- Any unlocked or unshared pages belonging to the virtual machine.
- Any of the unshared translation or backing store tables defining the address space of the virtual machine.
- Certain fields of the VMBLOK that are modified by routines that do not hold the system lock. Some of these fields are VMPSW, VMGPRS, and VMRSTAT.

The dispatcher obtains the VMBLOK lock before a virtual machine is dispatched and also before a CP request or an I/O request is unstacked. When a virtual machine is dispatched, the VMBLOK address of this virtual machine is saved in the processor's PSA in the field RUNUSER. Normally this virtual machine is also unlocked by the dispatcher when it is entered after an interrupt handler has finished processing. When RUNUSER is still locked, the PSA field LASTUSER is equal to RUNUSER. When RUNUSER is unlocked, LASTUSER is set to ASYSVM.

When a CP request or an I/O request is unstacked, the associated virtual machine is locked and the VMBLOK address is placed in register 11. When the dispatcher is entered after a CP request or an I/O request has been serviced, the virtual machine whose VMBLOK address is in register 11 is locked and will be unlocked by the dispatcher. This virtual machine may not be the same virtual machine that was locked when the CP request or the I/O request was unstacked.

A CP routine must lock another virtual machine for any of the following reasons:

- The routine, or a routine it calls, accesses any unshared page of the virtual machine.
- The routine, or a routine it calls, alters any field of the VMBLOK that is serialized only by the VMBLOK lock.
- The routine, or a routine it calls, could be interrupted and an exit taken to the dispatcher.

The original VMBLOK lock must be released before gaining the new lock.

Figure 31 on page 236 shows the modules that obtain the VMBLOK lock for a virtual machine other than the one requesting the service.

There are situations when a CP routine may access a virtual machine without locking it. If the CP routine, or any routine it calls, is only altering VMBLOK fields that are serialized by the system lock, locking the virtual machine is not necessary. For example, to process the SET PRIORITY command for a virtual machine, locking the virtual machine is not necessary since the altered VMBLOK field, VMUPRIOR, is serialized by the system lock. But to process the SET FAVORED command, locking the virtual machine is necessary since some of the VMBLOK fields altered, such as VMRSTAT, are only serialized by the VMBLOK lock.

DMKLOKFR	- Free Storage Lock
DMKLOKRL	- Run List Lock
DMKLOKTR	- Timer Request Queue Lock
DMKLOKDS	- Dispatcher Queues Lock

- CPEXBLOK Queue Lock

  deferred execution blocks

  processor related blocks

- IOBLOK/TRQBLOK Queue Lock

These are system spin locks that are held for very short periods of time. The control program code that runs without the global system lock must manipulate these queues and these locks insure system integrity along the unlocked paths.

User-Defined Locks

If you have user-defined areas that are used by more than one virtual machine and you need to serialize their use, you will need to define your own locking conventions. You can use the LOCK macro to obtain and release a PRIVATE lock. *VM/SP CP for System Programming* has details on how to code the LOCK macro.

Module	**Action**
DMKAPI	Locks the virtual machine that was last dispatched.
DMKBLD	Locks the virtual machine just built.
DMKCFO	Locks the virtual machine being set as favored.
DMKCNS	Locks the virtual machine associated with a real device block.
DMKCPS	Locks the virtual machine whose virtual device is being reset when a real device is halted.
DMKCPP	Locks each virtual machine in order to prepare the VMBLOK for uniprocessor mode.
DMKCPV	Locks the virtual machine whose storage is being locked or unlocked, or for whom accounting is being done.
DMKCSU	Locks the virtual machine receiving transferred spool files.
DMKDIA	Locks the virtual machine of the dialed system, the virtual machine of the line being dropped (DMKDIADR), or the virtual machine that owns the channel-to-channel device being coupled.
DMKGRF	Locks the virtual machine associated with a real device block.
DMKLOG	Locks the virtual machine being reconnected or the virtual machine being autologged.
DMKMID	Locks the virtual machines receiving messages at midnight.
DMKMSG	Locks the virtual machine receiving a message.
DMKMSW	Locks the system operator.
DMKNES	Locks each virtual machine active when the NETWORK SHUTDOWN command is processed.
DMKNLD	Locks the virtual machine associated with a real device block.
DMKPAG	Locks the virtual machine associated with a queued I/O request.
DMKPTR	Locks the virtual machine from which a page will be stolen.
DMKQCN	Locks the system operator.
DMKRGA	Locks the virtual machine associated with a NICBLOK.
DMKRGB	Locks the virtual machine associated with a NICBLOK.
DMKRNH	Locks the virtual machine of the destination user for a console task or the virtual machine associated with a remote teleprocessing line.
DMKSPL	Locks the virtual machine receiving a transferred spool file or the virtual machine owning a spooled reader file.
DMKVCA	Locks the virtual machine of the coupled-to CTC device.
DMKVCH	Locks the virtual machine to which the channel is being attached, or the virtual machine from which the channel is being detached.
DMKVDA	Locks the virtual machine involved in attaching or detaching a real device.
DMKVDD	Locks the virtual machine involved in detaching a real device.
DMKVMC	Locks the virtual machine to which the caller is communicating.

Figure 31. Modules that Obtain Additional VMBLOK Lock

Machine Check Handler -- Attached Processor and Multiprocessor Applications

A machine check interrupt is initially handled without the global system lock. DMKMCH determines if the error requires system termination, virtual machine termination, or simply recording and continuation. If the system was in a wait state or a virtual machine was in control and the system is not to be terminated, the machine check handler requests the global system lock with the defer option. If the lock can be obtained, normal DMKMCH processing continues. If the lock cannot be obtained, DMKMCH stacks a CPEXBLOK with CPMCHSE set and exits to DMKDSPRU. This CPEXBLOK causes processing to resume at DMKMCHSE with the global system lock held. Any machine checks that occur before the CPEXBLOK processing has completed are considered recursive machine checks and handled appropriately. If the control program was in control and the system is not to be terminated, the machine check handler saves status in the CPEXBLOK, set CPMCHSE and reloads MCOPSW. CPMCHSE is set to prevent the dispatcher from starting any new work on this processor until the machine check processing has completed.

DMKMCH passes control to DMKACRCT if channel termination has been set. DMKACR will attempt to recover the failing channel or channels by issuing CLRCH to each affected channel. If CLRCH does not restore a failing channel to an operational state, and if the system can continue operation without the failing channel, DMKACR will mark all paths through the channel offline and return to DMKMCH.

DMKMCH passes control to DMKMCTPT if the system is running in attached processor multiprocessor mode and a decision has been made to terminate the system. In general, if a virtual machine was running when the machine check occurred, only that virtual machine is terminated.

DMKMCTPT determines if the system can continue and if the processor can continue. If the machine check was not a clock error or the control program was in control on either processor, the other processor is signaled to stop and store status and a wait state PSW is loaded on the failing processor. An attempt is made to issue message 610W to the operator before the main processor is stopped. If the machine check was a clock error on a non-I/O processor and the control program was not in control, the main processor is signaled via an external call to initiate automatic processor recovery with an indicator to continue processing. If the error was a clock error on an I/O processor, and (a) the system is an MP system or (b) the channel-set switching facility is installed, and if the control program was not in control, automatic processor recovery will be initiated on the nonfailing processor.

The malfunction alert interrupt handler (DMKMCTMA) receives control from the external second level interrupt handler. If the malfunction alert came from the main processor in an AP system, and if the channel-set switching facility is not installed, a X'001' wait state disabled PSW is loaded. If the malfunction alert came from an attached processor and a virtual machine was in control, an indication is set to terminate the virtual user and CPAPRPND is set for processor recovery. If the attached processor was in supervisor state, message 610W is sent to the operator and a 013 wait state PSW is loaded. If the attached processor was in a wait state, CPAPRPND is set for processor recovery. If the malfunction alert occurs in a MP system and if a virtual machine was in control on the failing processor, an indication is set to terminate the virtual machine and automatic processor recovery is initiated.

The automatic processor recovery routine (DMKMCTPR) receives control from the external SLIH or the dispatcher. If the system is to continue processing, the vary processor offline routine (DMKCPPUP) is called. DMKCPPUP examines the chain of virtual machines for affinity to the failing processor and shared segment pointers. and shared segment pointers. Any shared segment pointers for the failing processor are switched to point to the recovery processor's shared segments. All the system control blocks and save areas necessary to run in attached processor or multiprocessor mode are also freed. The time from the first timer request queue element is placed into the clock comparator for the main processor.

While preserving the maintained fields in the absolute zero area, the recovery processor's prefix storage area is copied to the absolute zero area and prefixing is stopped. The APUOPER (or MPUOPER) in the PSA is turned off in the absolute zero area, and the prefix storage areas for the both processors are freed. The pages and DASD slots held by the failing processor for shared segments are freed by DMKPGT and DMKPTR. A message (194I) is issued, and return is made to DMKMCTPR. For any virtual machines with affinity to the failing processor, DMKMCTPR resets the affinity for each, issues message 621I, and puts the machine in console function mode (if the virtual machine is not disconnected). If a virtual machine is to be terminated, the virtual machine is reset, messages 616I and 619I issued. Normal return causes the system to continue processing in uniprocessor mode.

The action that the machine check handler takes for a given situation is determined by the error itself, the operating environment of VM/SP, and whether the system was performing a CP function or a virtual machine function -- or the system was not performing at all (a loaded wait state condition when the error occurred). Figure 32 on page 239 clarifies the action the system takes for the given situations.

Error Condition	VM/SP Processing				Virtual Machine Processing			
	Uni-Proc.	Attach. Proc.		Multiproc. Oper.	Uni-Proc.	Attach. Proc.		Multiproc. Oper.
		Main	Attach.	Either Proc.		Main	Attach.	Either Proc.
Invalid machine check interrupt code	1	1	1	1		1	1	1
Invalid PSW data	1	1	1	1	1	3	3	3
Register, Program mask instruction address invalid	1	1	1	1	1	3	3	3
System damages	1	1	1	1	1	3	3	3
TOD or CPU Clock Errors	1	1		1	1	1	1	3,4
Multibit (solid) Storage error	1	1	1	1	3,2	3,2	3,2	3,2
Multibit (intermittent) storage error	1	1	1	1	3,2	3,2	3,2	3,2
Storage Protect Key (solid) failure	1	1	1	1	3	3	3	3
Storage Protect (intermittent) failure	2	2	2	2	2	2	2	2
Malfunction alert	5	1	1	1	5	1	3,4	3,4
Channel inoperative	6	6	5	6	6	6	5	6

Legend	
1 =	load unit state PSW
2 =	refresh for retry operation
3 =	terminate the virtual machine
4 =	automatic processor recovery
5 =	Not applicable
6 =	Channel recovery

Figure 32. Condition/Action Table for Uncorrectable Errors

Multiprocessor External Interrupts

For external interrupts that can occur in attached processor/multiprocessor mode (time-of-day sync check, malfunction alert, external call, and emergency signal), DMKPSAEX gives control to DMKEXTSL. DMKEXTSL does the following for each kind of interrupt:

Malfunction alert

- Call DMKMCTMA, which will either load a disabled wait state on the appropriate processor or initiate automatic processor recovery, to allow the system to run in uniprocessor mode. If a user was running at the time of the malfunction alert he is terminated.

SHUTDOWN Emergency Signal

Issued prior to shutting the system down.

- Turn off APUOPER in each PSA to indicate that the system is in uniprocessor mode.
- Load a 008 disabled wait PSW on the receiving processor.

EXTEND Emergency Signal

- Disable channel zero.
- Pass control to the dispatcher at DMKDSPRU.

EXTEND EXIT Emergency Signal

- Enable channel zero in control register 2.

QUIESCE Emergency Signal

- Give control to the dispatcher at DMKDSPRU, which will load a wait PSW that is enabled for external calls only.

SYNC Emergency Signal

Issued by DMKCLKMP when the clocks are no longer synchronized (low order synchronization).

- Give control to DMKCLKAP to synchronize the clock on the attached processor. If the set clock fails, the nonIPLed processor is terminated with a CLK003 abend.

CLKCHK Emergency Signal

- Give control to DMKCLKCC. If the clock on the nonIPLed processor is not synchronized with the IPLed processor (high order synchronization) or is not set, then a flag is set to cause DMKCLKMP on the main processor to synchronize the clocks. The nonIPLed processor is then put in a wait state enabled for external interrupts. If the clock is not working, the nonIPLed processor is terminated with a CLK003 abend.

APR External Call

- Give control to DMKMCTPR to allow the system to run in uniprocessor mode.

RESUME External Call

Cancels a previous QUIESCE.

- Give control to the dispatcher at DMKDSPRU.

WAKEUP External Call

Wake-up an idle processor.

- If the system was running a user, reload the external old PSW.
- If the system was not running a user, then try to obtain the SYSTEM lock.
- If the SYSTEM lock is obtained, give control to the dispatcher at DMKDSPCH.
- If the lock is not obtained, give control to the dispatcher at DMKDSPRU.

DISPATCH External Call

Inform the other processor of a processor related CPEXBLOK.

- Try to obtain the global system lock.
- If the system lock is obtained, go to the dispatcher at DMKDSPCH.
- If the lock is not obtained and the system was in a wait state, go to DMKDSPRU.
- If the lock was not obtained and the system was not in a wait state, reload the external old PSW.

Time-of-Day SYNC Check

- Call DMKCLKSC. DMKCLKSC signals the nonIPL processor to quiesce. It then sends message DMKCLK970W to the operator and calls DMKCLKMP. DMKCLKMP issues a SYNC emergency signal to synchronize the clocks. DMKCLKSC issues a RESUME signal to allow the quiesced processor to continue.

- If the SYSTEM lock is held, go to the dispatcher at DMKDSPCH.
- If the SYSTEM lock is not held, go to the dispatcher at DMKDSPRU.

I/O Subsystem

Mainline, non-error processing in the I/O subsystem runs without the global system lock. Access to fields in the real I/O control blocks (RCHBLOK, RCUBLOK, and RDEVBLOK) is serialized by the I/O lock, a global spin lock. In an attached processor environment, only the main processor is capable of initiating I/O requests and receiving I/O interrupts. If the I/O subsystem running on the attached processor requires that I/O be started, a SWITCH will be issued to resume processing on the main processor.

In a multiprocessor environment, both processors have I/O capability. If either processor receives an I/O request, that processor attempts to initiate I/O operations.

At system generation time, when a channel path to a device is defined on one processor, an alternate logical path is automatically defined for the other processor. Thus, both processors *can* have access to any I/O device in the MP configuration.

If either processor receives an I/O request, that processor attempts to initiate the I/O operation on one of its own paths to the required device. If none of the online paths to the required device is available from the executing processor, that processor queues the I/O request on all busy and scheduled paths to the device, both its own and those of the other 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 for that processor.

While it is not required that both processors have access to all I/O devices, heavily used devices should be accessible by both processors to provide efficient system operation and to increase the possibility of system recovery following a processor or channel failure.

Shared Segment

The shared segment subfunction of VM/SP (DMKATS, DMKCFG, DMKCFH, DMKPGS, and DMKVMA) runs under the global system lock on either processor. All protected shared segments are duplicated in a system that is generated for attached processor or multiprocessor mode and that is initialized on a machine with the multiprocessing feature. DMKCFG obtains sufficient storage to construct the duplicate page and swap tables in contiguous storage. The SHRTABLE SHRPAGE pointer points to the page and swap tables for the main processor, and the page and swap tables for the attached processor are at a fixed displacement from the page and swap tables for the main processor. DMKCFG initializes both sets of page and swap tables. Initially, the two swap tables point to the DASD locations specified in DMKSNT. However, as the pages are read into storage and then stolen, each shared page is allocated its own DASD slot and is pointed to by only one swap table entry.

The last user to purge a shared system causes both sets of page and swap tables to be released.

One shared page table is reserved for use by each processor. This includes both problem state and supervisor state execution on behalf of a virtual machine. To accomplish this, each time a virtual machine running a shared system is locked, a test is made to determine whether or not the virtual machine was last serviced on this processor. If it was last serviced on the other processor, all of its shared page table pointers in its segment tables are switched to this processor's shared pages.

DMKPTR is able to steal a shared page from a shared page table reserved for the processor it is running on without notifying the other processor. The virtual page could not appear in the look-aside buffer of the other processor.

The dispatcher releases the VMBLOK lock on LASTUSER following the check for pending interrupts (assuming no fast redispatch possible) unless the virtual machine was running one or more shared systems. In the latter case the VMBLOK lock is not released until the DMKVMA scan for a changed page is completed.

DMKVMA scans all protected shared segments that the virtual machine used. For every changed page that it finds, DMKVMA checks whether or not the system lock is held. If the system lock is held, the changed page is returned to CP free storage. If the system lock is not held, DMKVMA marks the page table entry as invalid, marks the swap table entry as in transit, and indicates that the core table entry is on the free and flush lists. The other virtual machines can continue to use the shared segments. The changed pages are replaced when the next reference to the changed page is made.

If the shared segment is violated, an error message (DMKVMA456) is sent to the violator, and he is placed in console function mode. The user may examine his PSW and registers to determine what caused the violation. The user enters the BEGIN command to resume execution at the point of interruption.

Part 2: Method of Operation and Program Organization

This part contains the following information:

- CP Program Organization
  - Use of the Annotated Flow Diagram
  - VM/SP CP Interruption Processing
  - Virtual I/O Operations and Interruption Processes.

Chapter 7. CP Program Organization

Use of the Annotated Flow Diagram

The following text sections, which describe each major CP function, are annotated flow diagrams. These diagrams, consisting of logic labels and commentary, describe the general flow and use of CP logic modules and their relationship to other modules while performing a specific function or task. The annotated flow diagrams do not contain references to error messages, abnormal termination conditions, or most control block field labels. This avoids complexity and makes the general logic of CP and its related tasks more understandable to the user. With understandability as the key, obtuse and complex logic that is used for obscure and seldom used functions is not described. Also the flow diagram does not indicate nor describe every entry point encountered in a function. Nor do the diagrams illustrate the innumerable times that commonly used modules are utilized. DMKFRE and DMKCVT, the obtaining and returning of free storage and the number base conversion modules are such examples. Annotated flow diagrams are arranged by function and subfunction. Titles for these functions and subfunctions also precede annotated flow text and labels. The text in the charts is prefixed by underscored and capitalized entry points and labels. Entry points are indicated by seven or eight characters; the first three characters are DMK. Labels are indicated by prefixing with a comma and the six-character module identification.

The annotated flow diagrams in this part do not reflect VM/SP use of the MSS. If there is an MSS attached to the VM/SP system, consult Appendix B, "VM/SP MSS Support" on page 417 for flow diagrams of those functions that utilize the MSS (such as logging on a virtual machine that has a minidisk defined on an MSS 3330V volume).

Note: Annotated flow diagrams are not to be construed as trace material. The dynamics of CP operations preclude the use of the annotated flow diagrams, as they are shown in this manual, as traces of CP functions.

VM/SP CP Interruption Processing

SVC Interruptions - Problem State

DMKSVDIN
Entry for SVC interruptions from problem state. For problem mode and ADSTOP (SVC X 'B3'), the overlaid instruction is replaced.

DMKCFMBK
Console function mode is entered.

DMKSVDIN
For problem state SVC 76 (X'4C') check for valid parameter passing.

DMKVERD, DMKVERO
Determine the operating SCP used in the virtual machine by examining passed parameters in R0 and R1.

DMKSVD, SVCVER
For invalid parameter passing, error recording is not performed.

DMKSVD, REFSVCB
REFSVCB is called if TRACE SVC was in effect or if the virtual machine's page zero is not in real storage. Obtains the system lock before continuing. If the system lock is not immediately available, REFSVCB defers the interrupt and exits to DMKDSPRU.

DMKTRCSV
The DMKTRC module is called if TRACE SVC was invoked.

DMKPRGRF
If tracing is not active, flag user as being in instruction wait state and reflect the SVC back to the user.

DMKSVD
If the virtual machine's page zero is in real storage, generate and store an old SVC PSW. Fetch the new SVC PSW. If there is no PSW state change, store user's new PSW in RUNPSW, restore registers and dispatch via LPSW.

DMKSVD, REFSVCA
If there is a PSW state change, obtain the system lock before continuing. If the system lock is not immediately available, defer the interrupt and exit to DMKDSPRU.

DMKDSPB
Check the altered PSW.

SVC Interruptions - Supervisor State

DMKSVCIN
Entry for SVC interruptions from supervisor state.

DMKSVC, SVCDIE
Entry is for a system failure and is a SVC 0 or SVC 4 abend condition.

DMKDMPDK
Perform partial or full real storage dump.

DMKSVC, SVCLINK
Entry via SVC 8 provides linkage to a called routine in R15.

DMKPTRUL
If called routine is not resident, page it in and return control to the caller by loading the SAVERTN into the old PSW and then load the old PSW. The caller's addressability, SAVEAREA address and return address are maintained in a new SAVEAREA.

DMKSVC, SVCRET
Entry via SVC 12 return control from the called routine to the calling routine and restores addressability via R12 and R13.

DMKPTRUL
If a nonresident module, unlock page to return it to DASD.

DMKSVC, SVCRLSE
Entry via SVC 16 to release the current SAVEAREA used by SVC 8 and 12. Return to caller.

DMKSVC, SVCGET
Entry via SVC 20 to obtain a new SAVEAREA. Return to caller.

DMKSVC, SVCSWIT
Entry via SVC 24 to switch control to the main processor.

External and Clock Interruption Reflection

DMKPSAEX
Entered via the interruption key on system console, adjust accounting to charge for supervisor overhead. If problem mode, attention interruption, update the virtual machine PSW from the external old PSW.

DMKPSA, EXTBUTTN
Exit to dispatcher, if there is no logged-on operator, or the operator is disconnected, or there is no active terminal. If the operator was logged on and the external interruption key was pressed, disconnect the operator's terminal.

DMKQCOCL
Clear all console requests.

DMKSCNRD
If the device is a terminal or graphic device, issue HIO to the real device.

DMKDSPCH
Exit to the dispatcher.

DMKPSA, EXTBUTTN
For 37xx, convert resource identifier for the NCP terminal for the indexable entry into the NICBLOK for the associated VMBLOK, then

DMKRNHND
Reset all BTUs.

DMKDSPCH
Exit to the dispatcher.

DMKPSA, EXTEXTD
Upon location X'80' timer interruption, indicate the user end of the time slice by storing flag in the VMBLOK's VMOSTAT.

DMKDSPCH, DMKDSPRU
If the system lock is held or is available, exit to the main entry of the dispatcher, DMKDSPCH. Otherwise, exit to DMKDSPRU.

DMKPSA, EXTTIMER
Upon processor timer interruption, VMTLEVEL in VMBLOK as a real processor timer interruption.

DMKTMRVT
Simulate the interruption.

DMKDSPCH, DMKDSPRU
If the system lock is held or is available, exit to the main entry of the dispatcher, DMKDSPCH. Otherwise, exit to DMKDSPRU.

DMKPSA, EXTCKC
Upon clock comparator interruption reflection

DMKSCHTQ
Use the printer to unchain the active TRQBLOK. Call DMKSTKIO.

DMKSTKIO
Stack the block.

DMKDSPCH, DMKDSPRU
If the system lock is held or is available, exit to the main entry of the dispatcher, DMKDSPCH. Otherwise, exit to DMKDSPRU.

Monitor Interruption Processing

DMKMON

The VM Monitor data collection component uses both sample and trace techniques. Selected system counters are sampled by routines entered periodically via TRQBLOK. Selected events are traced upon execution via monitor call instructions embedded at strategic points in the control program.

DMKENTTI

Entered via TRQBLOK every two seconds (unless specified otherwise with the MONITOR INTERVAL command). A new TRQBLOK is immediately stacked via a call to DMKSCHST to specify return of control to the same entry point two seconds later. This subroutine is a high frequency (relative to the PERFORM, USER, DASTAP sampler) I/O status sampler. All channels are tested for a busy condition with a TCH instruction. All control units and devices are tested for a busy condition by examining the appropriate CP control blocks. The data obtained is accumulated for later sampling by the DASTAP class of data collection in a class 6 (DASTAP) code 2 (I/O status) record. The subroutine DMKENT62 performs this collection after the standard class 6 (DASTAP) code 1 record has been collected by MONCOD61 in DMKMONTI.

DMKMONMI

Entered from DMKPRG after a monitor call in a class currently enabled (as defined in CR 8 mask) has been executed by CP in supervisor state. The monitor call instruction number and code number stored by the hardware in the PSA are used to index branch tables to reach the appropriate data collection routines. As necessary, the data is stored in the monitor I/O buffers before output. Upon completion, control returns to instruction after monitor call.

Class	Code	Activity Being Monitored
1	0	Begin console read
	1	Console output
	2	End console read
	3	Console sleep
2	2	User dropped from queue
	3	User added to queue
	4	User added to eligible list
5	0	Privileged instruction being simulated
7	0	SIO for DASD SEEK
8	0	Add queue, drop queue - more detailed resource consumption data

DMKMONPR
All data collection subroutines use a common buffer management subroutine to obtain sufficient space in the monitor buffers. When not enough space is available, a switch is made to the next buffer in the chain and the full buffer is scheduled for output via a CPEXBLOK. I/O is handled by DMKIOSQR if tape is in use, or by DMKMIAWO if a spool file is in use. If data collection gets ahead of buffer output and all the monitor buffers are filled, a temporary suspension occurs.

DMKMONIO
Handles normal and abnormal completion of buffer output to disk or tape. For normal completion, the buffer used for I/O is made available for further data collection; if the next buffer is already full, its output is immediately scheduled. If a suspension was in effect, data collection is immediately resumed using the freed buffer. (*Note*: Suspensions should be eliminated by increasing the buffer allocation, using the MONITOR command or the SYSMON macro.) Special tape conditions that can be handled include end of tape and permanent error.

DMKENTKC
Entered via CPEXBLOK at midnight if automatic monitoring to spool file is in effect and it is required to close out the current file and continue monitoring with a new file. DMKENT satisfies the nucleus residency requirements of CPEXBLOK entry point and acts as a stepping stone to DMKMIA. Goes to DMKDSP after successful call to DMKMIAKC.

DMKMIAKC
Sets up a request to invoke a MONITOR CLOSE command in DMKMCCCL.

DMKMCCCL
Executes MONITOR CLOSE command and calls DMKMIACC to complete processing.

DMKMIACC
Invoked by the MONITOR CLOSE command to close the spool file and chain the spool file block to the reader of the virtual machine where data reduction is to take place. Starts new spool file if appropriate.

DMKENTST
Entered via TRQBLOK due to previous determination by automatic monitoring facilities that a MONITOR START SPOOL command should be issued. This entry satisfies the need for CP nucleus residency and immediately calls the pageable DMKMIAIN.

DMKMIAIN
Builds a message buffer containing a MONITOR START SPOOL command and calls DMKMCCCL.

DMKMCCCL
Executes MONITOR START SPOOL command. DMKENTST gives control to DMKDSP after successful execution.

DMKENTET
Entered via TRQBLOK due to previous determination by automatic monitoring facilities that a MONITOR STOP command should be issued at this time. This entry satisfies the need for CP nucleus residency and immediately calls the pageable DMKMIAEN.

DMKMIAEN
Builds a message buffer containing a MONITOR STOP command and calls DMKMCCCL.

DMKMCCCL
Executes MONITOR STOP command. DMKENTET gives control to the dispatcher after successful execution.

DMKMIAST
Entered from DMKCPI when it is determined that automatic monitoring has been requested via the SYSMON macro in DMKSYS and that TRQBLOKs should be queued via calls to DMKSCHST to invoke a MONITOR START SPOOL command and a MONITOR STOP command at specified times in the future. If monitoring is required to start immediately because the start time has passed, a CPEXBLOK is built to give control to DMKENTSC, which invokes the DMKMIAIN mechanism described above.

All other DMKMCC, DMKMNI, DMKMNJ, and DMKMIA entry points are used as a result of the processing of MONITOR commands or special conditions.

Three Class 0 monitor call codes have been reserved for special purposes. They are used without actually executing monitor calls, but as a result of MONITOR command processing. They are:

Class	Code	Function
0	97	Write header record after MONITOR START command
	98	Write trailer record after MONITOR STOP command
	99	Write suspension record when data collection resumes

Program Interruption Processing

DMKPRGIN
For a program interruption received while in supervisor mode (indication of CP module error) and INTRDR+1 does not indicate MONITOR CALL (X'40') exit to -

DMKPRG, CPERROR
Send abend message to the system operator.

DMKDMKPK
Dump storage and initiate loading (via IPL).

DMKPRGIN
For supervisor state and MONITOR CALL save registers in DMKPRGPR.

DMKPRGMI
Do MONITOR CALL interruption processing (DMKMON).

DMKPRG, PRNSTAT
For PER Interrupt (X'80'), save PER event information (PER code and PER address) in the PERBLOK (if CP PER) or the ECBLOK (if user PER) and turn on the PER pending flag (VMPERPND).

DMKPRG, PRNSTAT
For paging exception X'11' and EC mode with translation on call DMKVATEX.

DMKVATEX
Process the exception.

DMKPRGIM
For paging exception X'11' and EC mode with translation off, and enabled for I/O interrupts and PAGEX on call DMKVATPF.

DMKVATPF
Process the pseudo page fault.

DMKPRG, OBSLOCK
For all other page fault conditions go to DMKPTRAN.

DMKPTRAN
The system lock must be obtained before DMKPTRAN is called. If the system lock is not immediately available, defer the interrupt and exit to DMKDSPRU.

DMKPTRAN
Bring in the page from the auxiliary device.

DMKDSPCH
Exit to dispatcher.

DMKPRG, PRNSTAT
For segment exception X'10' with EC mode on and translation on call DMKVATSX.

DMKVATSX
Process the exception.

DMKPRG, PRGSIMI
For the segment exception, X'10' does not follow the above parameters; process it as an addressing exception.

DMKPRG, TRANSEX
Process X'12' translation exceptions.

DMKPRG, PROG01
For a privileged operation exception of a virtual machine in supervisor mode, examine INTPR+1. If X'02', call DMKFPS; otherwise, call DMKPRVLG.

DMKFPS
Process the exception and, if successful, dispatch the user. If unsuccessful, return to DMKPRGIN.

DMKPRVLG
Process the exception.

DMKPRV, DMKPRGSM
For virtual machines in problem mode, store the users new program PSW in VMBLOK VMPSW.

DMKPSASV
When the program interrupt occurs and the users page 0 is not resident or the virtual machine is in EC mode, paging is performed.

DMKDSPB
Check the new PSW.

DMKPRVLG
Validate the privileged operation indicated in VMINST and perform the service.

Code	**Operation**
X'08'	SSK - Set storage key
X'09'	ISK - Insert storage key
X'44'	EX - Execute instruction
X'80'	SSM - Set system mask
X'82'	LPSW - Load PSW
X'9C'	SIO - Start I/O
X'9D'	TIO - Test I/O
X'9E'	HIO - Halt I/O
X'9F'	TCH - Test Channel
X'AC'	STNSM - Store, then AND system mask
X'AD'	STOSM - Store, then OR system mask
X'AE'	SIGP - Signal processor
X'B1'	LRA - Load real address
X'B202'	STIDP - Store processor ID
X'B203'	STIDC - Store channel ID
X'B204'	SCK - Set TOD clock
X'B206'	SCKC - Set TOD clock comparator
X'B207'	STCKC - Store TOD clock comparator
X'B208'	SPT - Set CPU timer
X'B209'	STPT - Store CPU timer
X'B20A'	SPKA - Set PSW key from address
X'B20B'	IPK - Insert PSW key
X'B20D'	PTLB - Purge TLB
X'B210'	SPX - Set prefix
X'B211'	STPX - Store prefix
X'B212'	STAP - Store CPU address
X'B213'	RRB - Reset reference bit
X'B221'	IPTE - Invalidate Page Table Entry
X'B22C'	TB - Test Block
X'E501'	TPROT - Test Protection
X'B6'	STCTL - Store control registers
X'B7'	LCTL - Load control registers
X'BA'	CS - Compare and swap
X'BB'	CDS - Compare double and swap

DMKPRV, LOCKET

The system lock must be obtained before other supervisor routines are called. If the system lock is not immediately available, defer the interrupt and exit to DMKDSPRU.

DMKHVCAL

On privileged operations of DIAGNOSE X'83' and the associated function code, perform the service.

DMKVSIEX

Execute privileged I/O operations of SIO, HIO, TIO, and TCH.

DMKTMRTN

Perform privileged operations related to TOD clock, TOD clock comparator and the processor timer.

DMKPRGSM
Program interruption is reflected back to the user on invalid instruction operands, unsupported instruction operand codes and DIAGNOSE X'83' function codes that are not a multiple of 4.

Virtual I/O Operations and Interruption Processes

CTC Device Operations Between Two Virtual Machines

DMKVSIEX
Virtual I/O operation is reflected to DMKVCA, the channel device module, for processing.

DMKVCAST
For SIO, check if the CTC device (channel-to-channel adapter or 3088) is coupled. If not coupled, call DMKDIBSM.

DMKDIBSM
Simulate return status.

DMKVCA, VCRSTART
For a coupled CTC device, analyze operations resulting in X-side (read) and Y-side (write) of the data transfer operation.

DMKVCA, VCASIOB
Detected interruptions are presented to users via stacked IOBLOKs and DMKSTKIO.

DMKVCBTS
CTC device TIO activity is determined by examining Y-side information to determine mode and activity.

DMKVCBSH
CTC device HIO and HDV is processed by determining the condition code to present and whether the Y-side should be notified.

DMKVCBRD
CTC device process results from RESET xxx or SYSTEM RESET commands. The CTC device status is reset but the CTC devices are not uncoupled.

DMKVCBRS
Uncoupling CTC device is achieved in the VDEVBLOK (VDEVNRDY flag) idle CTC device plus an invoked DETACH xxx or user LOGOFF. Return to calling routine.

Scheduling I/O for CP and the Virtual Machine

DMKIOSQR
Entered via SVC. Entry point indicate a CP I/O event as indicated in the IOBLOK. For start request, increment the SIO count in the RDEVBLOK and start the device if it is available. If not (device busy or already scheduled) queue the IOBLOK and return the operation to the caller.

DMKIOSQV
Entered via SVC. Entry point indicates virtual machine initiated I/O event. Preserve VMBLOK address in R11, turn off IOBCP bit in the IOBLOK, add 1 to SIO count in the VDEVBLOK (or RDEVBLOK). Process the SIO if there is any available path to the device. If not, queue the IOBLOK and return the operation to the caller.

Standard DASD I/O Initiated via Diagnose

DMKDGDDK
Perform simple count-key-data disk I/O of a standard format. Entry is via DMKHVC code X'18'.

DMKSCNVU
Find device related to SIO cuu address.

DMKFREE
Allocate storage for IOBLOK and RCWTASK.

DMKDGDDK
Build and check the CCW string.

DMKIOSQV
Execute I/O. On completion, post condition code (and error return code in R15, if detected).

DMKDSPCH
Exit to dispatcher.

General I/O Operation Initiated Via Diagnose

DMKGIOEX
Perform general I/O operation. Entry is via DMKHVC code 20.

DMKSCNVU
Find device related to SIO cuu address.

DMKFREE
Allocate storage for the IOBLOK.

DMKCCWTR
Build the read CCW list.

DMKIOSQV
Queue the I/O request for execution.

DMKGIO, DIAGRIN
On interruption return, check status.

DMKUNTER
If no problem encountered, free storage used for CCW string and IOBLOK.

DMKGIO, DIAGRIN
Reflect the condition code and return code to the user.

DMKDSPCH
Exit to dispatcher.

DMKUNTRN
On returned error condition, convert real CSW to virtual CSW and set in user's page 0.

DMKGIO, GIOEXT
Exit via SVC 12.

Virtual Machine I/O Instruction Simulation and Interruption Reflection

I/O Instruction Simulation

DMKVSIEX
Entry from DMKPRV to simulate I/O per VMBLOK's VMINST field.

DMKVSI, VIOSIO
On detected SIO, call -

DMKSCNVU
To locate VCHBLOK, VCUBLOK, and VDEVBLOK for the cuu called per SIO instruction.

DMKVSIEX
Determine device availability and set condition code accordingly.

DMKIOSQV
If the operation is warranted, schedule the operation.

DMKVSI, VIOTIO
For TIO, check device status, pending interrupts, and set appropriate condition codes.

DMKVSI, VIOHIO
For HIO, check for dedicated channel, CE, CU, or device busy condition, and subchannel busy and set appropriate condition codes.

DMKVSI, VIOTCH
Check for dedicated selector or busy channel and check for pending abnormal interruption and set appropriate condition code.

Interruption Reflection

DMKVIOIN
Entry from DMKDSP to process the reflected virtual interruption.

DMKSCNVU
Locate the VCHBLOK, VCUBLOK, and VDEVBLOK.

DMKVIOIN
Analyze blocks and reflect condition code to user. If condition code equals 1 (CC=1), save status from the real device (if real device) and DMKUNTFR.

DMKUNTER
Translate and store CSW in user's page 0.

DMKVIO, VIOCC1
On TIO or HIO, free the device and set CC=1.

DMKFRET
Fret storage for the IOBLOK.

DMKDSPCH
Exit to dispatcher.

Virtual Console Simulation

DMKVSIEX
Entry for virtual console activity comes from the SCP stored in the user's virtual machine. The program's generated CCWs and data are reflected to the attached terminal used by the virtual machine operator.

DMKVCNEX
Locate and move non-TIC CCWs from the users virtual storage to a VCONCTL block.

DMKVCN, GETCCW
Update CAW and CSW in respective control block.

DMKVCN, VCNRD
For read operation, build a read console buffer VCONBUF for the input to be read from the terminal.

DMKQCORD
Queue a console read request.

DMKVCNEX
Set return address in VCONCTL VCNRDRET field.

DMKVSTCP
Spool console activity if SPOOL CONSOLE START specified.

DMKDSPCH
Exit to dispatcher. Wait for completion.

DMKVCN, VCNWR
Calculate and obtain free storage (VCONBUF) necessary for the write to console operation.

DMKVCN, VCNMDAT
Translate and bring in user's data page and move it into VCONBUF.

DMKQCNWT
Queue a console write request.

DMKDSPCH
Exit to dispatcher.

DMKVCN, VCNSNCN
ON a sense operation, set CE and DE in the virtual PSW. Reflect the PCI flag in the PSW if the PCI flag was set in the CCW. Set the IL flag if warranted. Move the sense data from the VDEVBLOK to user storage as designated by the CCW. Update VDEVBLOK's VDEVCSW to reflect status and count.

DMKVCN, VCNCC1
On completion of I/O operation, set appropriate status for command reject, not ready protection check, incorrect length, channel program check. Set appropriate CC and CSW in users page 0. Otherwise post pending interruption status in VMBLOK, VCHBLOK, VCUBLOK, and VDEVBLOK.

DMKVCN, FLAGTEST
If command chaining, process the next CCW.

DMKDSPCH
Exit to dispatcher.

Local Graphic I/O and Interruption Processing

DMKGRFEN
Entry for local graphic device enable and disable function (from DMKCPVEN and unstacked CPEXBLOK). Invoking CP ENABLE/DISABLE commands, start or terminate local 3270 display (and supported print devices) and certain system console activity.

DMKGRF, LOGUSER
Format and write out the logo at the screen.

DMKGRF, ATTNINT
Unsolicited attention for RDEVBLOK (enabled).

DMKBLDVM
Build LOGON VMBLOK for logon process.

DMKIOSQR
Schedule request to clear and display the logo.

DMKDSPCH
Exit to dispatcher to wait for interruption. Successful logon per the next interruption begins the operation of building the user's virtual machine.

DMKSCNRU
From the IOBLOK, locate the real device blocks related to the interruption. Analyze IOBLOK CSW and condition code and the I/O operation to determine read/write sequential action. For unit error, retry 10 times (if applicable). If recovery fails, log off. For ATTN interruptions, attempt to log on the new user if unsolicited ATTN occurs. Otherwise, set up for READ CCW string.

DMKIOSQR
Issue the SIO.

DMKDSPCH
Wait for the response.

DMKGRFIN
Local 3270 display and certain system console interruption entry from dispatcher. On response of CE and DE, go to auxiliary processing routine address in TRQBLOK extension TRQBCRT and execute the processing routines

DMKGRF, RDATA
Process read response of data plus ENTER key.

DMKCNSED
Edit and modify length count. Move data to caller's buffer.

DMKQCNWT
Schedule rewrite to screen (unless inhibited).

DMKIOSQR
Perform start I/O.

DMKDSPCH
Exit to dispatcher.

DMKGRFIC
Entry point to process CONTASKS queue for local 3270 devices.

DMKGRHIN
Entry point to build channel programs for 3066 devices.

DMKFREE
Get storage for IOBLOK and TRQBLOK.

DMKGRF, BLDCCWS
Execute CONTASK, if appropriate. If not -

DMKDSPCH
Exit to dispatcher.

DMKGRF, RDMINT
For read return, determine function key action and write response (if appropriate) via KEYTBL.

DMKGRFTI
Entry point for processing timer interrupts.

DMKGRCUP
Generate the 3270 orders required to update the screen status, clear the input area, or clear the output area.

DMKGRAQT
Perform APL/TEXT translation for outbound data.

DMKGRHIN
Entry point to handle 3066 CCW strings.

Locate and Validate an ISAM Read Sequence

DMKISMTR
Entry from DMKCCW modules to locate and modify an ISAM CCW string. Using the IOBLOKs IOBCAW locate the RCWTASK. Check for the ISAM read CCW.

DMKISM, CHKRD
Check for the correct ISAM sequence as follows:

1. The last CCW in the RCWTASK is a TIC.
2. This RCWTASK points to the next RCWTASK with a minimum of 2 CCWs.
3. The first modified CCW is in real storage.
4. The last byte of the ISAM read overlays the operation code of the first CCW in the next RCWTASK.
5. The TIC in the RCWTASK is to the next RCWTASK's first CCW.
6. The date address of the first CCW in the next RCWTASK is the same address of the ISAM read + 1 as it is in real storage.

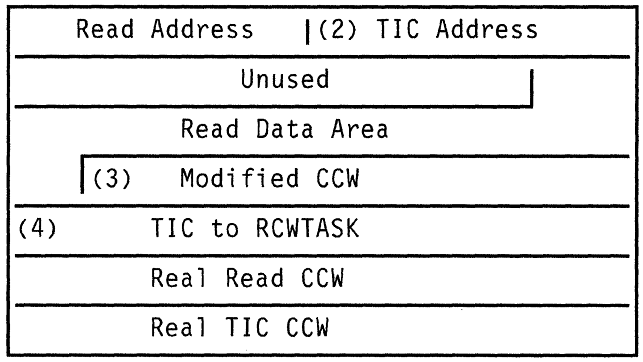
DMKFREE
Storage obtained for seven double words save block.

DMKISM, CHKTSK2
Institute the ISAM read modification as follows:

1. Set the read to point to the save block data area.

2. Set the CP TIC to point to the modified CCW in the same block.
3. Set the modified CCW (seek head) in the save block to point to the save block data area.
4. Set the CP TIC in the save block to return to the RCWTASK following the modified (seek head) CCW.
5. Set the search CCW in the RCWTASK to point to the data area in the same block.

DOUBLEWORD SAVE BLOCK



DMKISM, CHKTSK2
Return to DMKCCW module via SVC 12.

Scheduling CP and Virtual Machine I/O Operations and Interruption Handling

DMKIOSQR
Entry to process CP generated I/O. Flag the IOBLOK as a CP generated event. Initiate I/O if path to real device is free (available). If not, queue the IOBLOK and return to caller.

DMKIOSQV
Entry to process I/O for virtual machine I/O operations. Mark IOBLOK as not CP initiated. Save VMBLOK address. If path to the RDEVBLOK or the RCUBLOK is busy queue the IOBLOK and return to caller.

DMKIOS, IOSTRTDV
If available status, start the I/O and return to caller.

SIO Operations

DMKIOS, IOSTART
If I/O request has not been reset, save the address of the active IOBLOK and set device busy. If the device is being reset, unflag scheduled device and scheduled control unit. Stack the IOBLOK and restart the device.

DMKIOS, IOSSIO
Set the subchannel path busy and chain the active IOBLOK from the RDEVBLOK.

DMKIOS, IOSSIO

Locate caller's CAW and issue the SIO. Check SIO completion. Returned condition code sets sequel action.

CC = 0 indicates successful start
CC = 1 CCW stored, initiate sense operation
CC = 2 Busy condition, retry or requeue IOBLOK
CC = 3 Fatal error (not operational), stack the IOBLOK and return to caller

HIO Operations

DMKIOSHA

Entry point for halting a device. If device is not active, return to caller. If IOBLOK active, reset the IOBLOK to halt the device and mark the device reset in RDEVBLOK.

DMKIOS, IOS1OK

If the channel path is busy with a burst mode operation, stack the IOBLOK to halt the operation when the channel path becomes available. Return to caller.

Missing Interrupt Processing

DMKDID

Receives control from a timer interrupt (TRQBLOK).

DMKDIDDA

Checks for DASD devices when a missing interrupt is detected.

DMKDIDGR

Checks for graphic devices, except 1053 and 328X printers, when a missing interrupt is detected.

DMKDIDTA

Checks for tape devices when a missing interrupt is detected.

DMKDIDUR

Checks for unit record devices, except 3800 and 3289E printers, when a missing interrupt is detected.

DMKDIDMS

Checks for miscellaneous devices when a missing interrupt is detected.

DMKDIDLF

Cleans up certain missing interrupts at LOGOFF/FORCE time.

DMKDID, SCAN

Scans RDEVBLOKs. If RDEVBUZY is on, the RDEVTYPC field is examined to verify that the RDEVBUZY flag and timer interrupt are for the same class of device. If the classes match, the RDEVMID bit is turned on. If not, the scan continues. The timer is reset.

DMKDID
Stacks a CPEXBLOK to regain control.

DMKDID, DELQDV
Receives control from the CPEXBLOK and calls DMKIOSHA to perform a HALT DEVICE and CLEAR I/O.

DMKDID
Simulates I/O interrupt through DMKIOTRC. Schedules a ten second timer interrupt to check RDEVMID. A message is sent to the operator and a record is written to the system log record.

Interruption Processing

DMKIOTIN
Entry from I/O new PSW. Check old PSW. If problem mode, save processor status in the VMBLOK.

DMKSCNRN
Locate RCHBLOK, RCUBLOK, and RDEVBLOKs for interruption unit.

DMKVIOIN
Process dedicated channel interruption condition. If control unit end or channel available interruption occurs, restart the operation, if interruption does not occur stack it.

DMKIOTIN
If the IOBLOK is not active on RDEVBLOK interruption, call IOTGTIOB to construct an IOBLOK and continue.

DMKIOT, DOSENSE
Schedule sense operation, then go to dispatcher.

DMKIOS, COPYIOB
For PCI or CE interruptions, copy and stack the IOBLOK.

DMKCNSIN
Process PCI or CE interruptions, if related to local graphic device or nondedicated TP line.

DMKIOS, IOSCCOB
For split seek complete interrupt, rechain the seek and reschedule operations.

DMKSTKIO
Stack IOBLOK and restart any units freed by the interruptions.

DMKDSPCH, DMKDSPA
If the system lock is held or is available, exit to the main entry of the dispatcher, DMKDSPCH. Otherwise, exit to DMKDSPA to try to redispatch RUNUSER.

Terminal Console I/O Control, START/STOP, 3210, 3215, and Others

Enabling/Disabling

DMKCNSEN
Per unstacked CPEXBLOK, on enable or disable function, check current status of the current real device and set flag in RDEVFLAG. Build CONTASK and IOBLOK.

DMKIOSQR
Issue SIO for enabling or disabling function and check return.

DMKDSPCH
Exit to dispatcher.

Process CONTASK data

DMKCNSIC
Entry from DMKQCN module. Build I/O CCW string as defined by the console device type. Also select the proper line code to interface with the device. Place in CONTASK. For output CONTASK determine the correct translation table applicable to terminal communications (DMKTBL). To append proper control character to the data stream for the particular device type, refer to the following labels:

- DMKCNS, INCWTTY - Teletypewriters
- DMKCNS, INC2741 - 2741, 3767
- DMKCNS, INC1050 - 1050, 1051
- DMKCNS, INC3210 - 3210, 3215

DMKCNS, INCFINS
Attempt to start I/O by halting the current operation, if the operation is a `Prepare` CCW or the input is a read and the forthcoming output is a priority write CONTASK.

DMKFREE
Get storage to build IOBLOK, if needed.

DMKCNSIN
Set return address in IOBIRA.

DMKIOSQR
Start I/O. If busy condition encountered build CPEXBLOK and queue for later execution.

DMKDSPCH
Exit to dispatcher.

Start/Stop Terminal Interruption Process

DMKCNSIN, CMBREAK
For an active input task halted, RDEVFLAG=RDEVHIO to process priority output task.

DMKFREE
Build CONTASK for reverse break CCWs.

DMKCNS, CNSBREAK
Move the input CONTASK following the last priority write output CONTASK on the chain.

DMKCNS, CNSIOUC
For unit check with intervention required, assume an attention interruption and build a `prepare` CCW for the 2741.

DMKCNS, CNSLOGE
For unit check and timeout condition - logoff the virtual machine and re-enable the line.

DMKCNS, CNSRTRY
For data check and other conditions, retry the previous operation.

DMKQCOET
Process completed output CONTASK.

DMKCNSIN
Interpret interruption status and CCW residual count for input CONTASK completion.

DMKCNS, CNINCT
Validate input data and control characters and translate to EBCDIC from line code.

DMKTRMID
Attempt to identify, if applicable, the line code identification; PTTC/EBCD or correspondence.

DMKCNSED
Perform line editing of the input buffer.

DMKCNS, CNSRT41
Prepare and issue control CCWs to request status information from the terminal.

Processing the Control CONTASK Interruption

DMKCNSIN, CNSCTAK
For control task interruption return, examine the interruption status according to control task function:

- DMKCNS, CNSTAK - Reset control task
- DMKCNS, CNSCTID - Device identification
- DMKCNS, CNSCTPR - Attention signal

DMKCNS, CNSCPTR
Write `VM/370 Online` interpretation of response determines retry, or build new CONTASK and execute or stack or process next CONTASK.

DMKQCOET
Process completed CONTASK requests. If no tasks remain for the terminal, set IOBLOK's IOBIRA to DMKCNSIN and link the IOBLOK to the user.

DMKDSPCH
Exit to dispatcher.

Console Scheduling

DMKQCORD
SVC entry to build CONTASK for input data. Set the input buffer to zeros.

DMKFREE
Get storage to build CONTASK.

DMKQCONQ
Stack CONTASK on RDEVBLOK, if RDEVCON was zero. If not, exit to the appropriate interrupt handler per RDEVTYPC and RDEVTYPE or -

DMKDSPCH
Exit to dispatcher.

DMKQCNWT
SVC entry to build CONTASK for output data. Strip trailing blanks from output message, modify byte count and determine real device destination.

DMKFREE
Get storage to build output CONTASK.

DMKQCN, WRDSCK
Update CONTASK CCW message byte count for the message text, terminal and line control information and (if appropriate) time stamp.

DMKCVTDT
If time stamp required, get the value for CONDATA area.

DMKVSPVP,
Spool console message, if VDEVFLAG=VDEVCSPL.

DMKQCN, CRSCAN1
If message data contains carriage returns, X'15', create a separate CONTASK for each line.

DMKQCN, CHKCHAIN
For local and remote 3270s that are not accessed via VSCS, compute whether the 3270 screen will be filled. If not, operate as though the NORET parameter were specified and return to the caller so that the caller has an opportunity to generate more output lines.

DMKQCONQ, WAKEUPR
On first CONTASK or priority CONTASK, enqueue on chain from RDEVBLOK in appropriate location, then call related interrupt handler.

DMKQCONQ, WAKEMUP
If NORET or DFRET specified, build and stack CPEXBLOK to alert the interruption handler and return via EXIT SVC otherwise go to specified interruption handler.

DMKQCPTO
Entry via SVC to disconnect and logoff a virtual machine as a result of transmission line failures. Place the virtual machine in a wait state, VMRSTAT=VMCFWAIT.

DMKSCHDL
Alter virtual machine to unrunnable state.

DMKFREE
Get storage for message for the system operator.

DMKSCNRN, DMKSCNRD, DMKCVTBH, DMKSYSNM
Fill in message variables.

DMKSCNR, DMKSCNRD, DMKCVTBH, DMKSYSNM
Fill in message variables.

DMKQCNWT
Send the user disconnect message to the operator.

DMKQCP, DSCGTRQ
Build TRQBLOK, if needed, for 15 minute delay, schedule it, and exit via SVC.

DMKQCP,DSCTLOG
After time elapse, TRQBLOK is unstacked and VMOSTAT is set to VMKILL for inevitable DMKUSOFF logoff operation.

DMKDSPCH
Exit to dispatcher.

3704/3705 Interruption Handler

DMKRNHIC
Entry via DMKQCN or via CPEXBLOK for 3704/3705 resource initialization. Locate the NICBLOK and check resource availability.

DMKRNH, LINEBRK
For resource unavailable, set RC=12 in CONTASK save area and return task via DMKQCNET.

DMKRNH, TAGTASK
For resource available, set CONTASK values per input and output task requirements.

DMKRNH, TASENQ
Move CONTASK from RDEVBLOK chain to NICBLOK chain.

DMKRNH, RNSTART
On 3704/3705 available condition, search NICLIST and build an IOBLOK if required.

DMKRNHIC, RNEXLST
Search the NICBLOKs for CONTASKs to be sent to 3704/3705, build and chain for output.

DMKRNH, RNCHAIN
Perform necessary function for each resource.

DMKIOSQR
Start output I/O operations.

DMKRNH, RNICHNI
Return via R7.

DMKRNHND
Entry via SVC to schedule resource control tasks.

DMKRNH, RNHNDTK
Build control CONTASK and enqueue it for execution.

DMKRNH, STKCPEXT
For NORET specified, build and stack a CPEXBLOK to perform SVC exit.

DMKRNH, RNDEXIT
Attempt to start output via GOTO DMKRNHIC.

DMKRNH, RNFDISC
Entry for 3704/3705 recovery.

DMKNLDR
Load the 3704/3705, if it was not previously loaded.

DMKFREE
Get storage to build CKPBLOK (telecommunications control block), if necessary.

DMKRNH, RNSBITS
Record active line and enabled terminal flag bits.

DMKQCOET
Clear CONTASK chains.

DMKQCPTO
Force disconnect to all active users.

DMKNLEMP
DUMP the 3704/3705.

DMKNLDR
Reload the named program.

DMKRNHND
On `IPL complete` signal, re-enable resources.

DMKFRET
Release the CPEXBLOK.

DMKDSPCH
Exit to dispatcher.

DMKRNHIN
Entry via IOBLOK to perform input and output interruption processing.

DMKRNK, RNIOERR
For input process failure. Analyze the failure and if related to the 3704/3705 and not to a particular resource, either retry or dump and reload.

DMKRNH, READBUF
Interpret response codes for each BTU received and schedule necessary control operations.

DMKRNH, CMPREAD
Generate response to a read error.

DMKRNH, CMPWRITE
Generate response to a write error.

DMKRNH, CMPCONT
Generate response to a contact task error.

DMKRNH, CMPDISC
Generate response to a disconnect task error.

DMKRNH, COMCNTL
Generate response to a control task error.

DMKRNH, USOLIT
Generate response to a unsolicited read.

DMKQCOET
Return completed CONTASKs.

DMKRNH, RNSTART
Attempt to restart the 3704/3705.

DMKDSPCH
Exit to the dispatcher.

DMKRNHIN
Entry via IOBLOK to perform input and output interruption processing.

DMKRNH, SCHREAD
On output, examine interrupt status per IOBLOK values and if ATTN, build and start a read CCW sequence.

DMKRNH, RNIOEUC
If unit check and fatal, dump and reload the 3704/3705.

DMKRNH, RNOREAD
If pending ATTN cleared via SIO -

DMKIOSQR
Reschedule write operations.

DMKRNH, RNDLOWDN
If unit exception, set RDEVSLOW and reschedule rejected CONTASKs.

DMKQCOET
Return only CONTASKs without CONRESP or CONSPLT set. Retain others until final response is received.

DMKRNH, RNSTART
Attempt to restart the 3704/3705.

DMKDSPCH
Exit to dispatcher.

Handling Remote 3270 with Binary Synchronous Lines

Remote Display Station and Binary Synchronous Line Enabling/Disabling

DMKRGBEN
Entered when the NETWORK ENABLE/DISABLE command is issued.

DMKFREE
Get storage for the necessary CONTASK, IOBLOK, and if applicable, BSCBLOK.

DMKRGB, LINESUP
Set up required CCWs and control data in the CONTASK for tasks. These tasks include: enabling the binary synchronous line, READ PARTITION (QUERY) processing if appropriate, enabling a device, LOGO messages, screen formatting, and disable line or device (logoff).

DMKFREE
For logon function build logon VMBLOK.

DMKIOSQR
Start line I/O or device I/O, for not busy condition.

DMKRGB, RGFTASK
For busy condition, build CPEXBLOK and exit to caller.

Request Handler for 3270 I/O Events

DMKRGBIC
Entry from DMKDSP. On a not available line condition, exit to dispatch. For available line, process the associated CONTASKs by queueing the related resource from the NICBLOK.

DMKRGB, RGSTART
Process POLL SIO on a no CONTASK queued condition.

DMKIOSQR
Process selection SIO on available resources and not in control mode per NICBLOK conditions and the CONTASK CONSTAT field.

DMKDSPCH
Exit to dispatcher.

DMKGRAOT
Perform APL/TEXT translation for outbound data.

DMKGRCUP
Generate the 3270 orders required to update the screen status, clear the input area, or clear the output area.

Secondary Interruption Processor for 3270

DMKRGAIN
Entry from DMKIOS, examine line interruption condition. Discard any of the following and go to the dispatcher: non-binary synchronous line, copied IOBLOK, unsolicited interruption, bisync line flagged not-in-use, non-terminal class device.

DMKRGA, FATALER
For IOBFATAL condition or any non-zero condition code, free all related CONTASK, IOBLOK, IOERBLOK, and BSCBLOK.

DMKRGA, DISASTA
Log off all affected users on that line.

DMKMSWR
Send message to the system operator.

DMKDSPCH
Exit to dispatcher.

DMKRGAIN
If line or terminal response did not fall in the previous category, process via TP code branch. The code in the fifth byte of the ending CCW or IOBCSW-8.

DMKRGC
Process the input line.

DMKDSPCH
Exit to the dispatcher.

3270 Binary Synchronous Line Error Recovery

DMKBSCER
Entry via DMKIOS to process errors related to the binary synchronous line unit check and channel error conditions. On first error pass, move the IOERBLOK pointer from the IOBLOK to the RDEVBLOK, reset retry and fatal flags, set the ERP flag and call DMKFREE.

DMKFREE
Get free storage for a work area for retry CCWs.

DMKBSC, NOTFIRST
On a not first error condition, test for unrecoverable error condition. Unrecoverable errors include: program check, protection check, chaining check, equipment check, interface control check and channel control checks. If one of these, notify the system operator. Reset flags, initiate error recording and

DMKFREE
IOERBLOK.

DMKIOSQR
Go back to scheduler.

DMKRGA
Analyze TP code, sense data CSW residual count and retry count to determine retry or IOBFATAL flag setting.

Real Storage Allocation and Page Management

Process a Page Request

DMKPTRAN
Enter via the TRANS MACRO per paging request as determined by DAT created program interrupt (page or segment exception).

DMKPTR, RESTART
Return to caller, if virtual address in R1 is beyond range of user's directory specified storage size.

DMKPTR, ADDROK
Check page residency via LRA (LOAD REAL ADDRESS) operation.

DMKPTR, TESTLOCK
For resident page, lock page in storage (if appropriate).

DMKPTR, GETRADD
Set real address in R2, make PAGTABLE entry valid. Set CC=0 and exit to caller.

DMKPTR, INTRAN
For page not resident but in transit (SWPTABLE, SWPFLAG), place virtual machine in locate mode. Locate CPEXBLOK for the real page requested and chain another CPEXBLOK with a return address of TRANRETN, to the same chain.

DMKPTR, TRANRETN
After page is no longer in transit, restore registers and return to RESTART for processing.

DMKPTR, GETPAGE
Reclaims a page on FREELIST (CORETABLE).

DMKPTR, DOIO
For page that is not in storage, do setup to read in the page.

DMKPTR, CKDEFER
For DEFER option passed in R2, build CPEXBLOK to return to user after page is in storage.

DMKPTR, PAGIN
After the page is read into storage DMKPAGIO process, remove the user from the wait state and update the lock count (if required).

DMKPTR, GETRADD
Set real address in R2, make PAGTABLE entry valid. Set CC=0 and exit to caller.

Obtain, Return, Lock and Unlock a Page of Free Storage

DMKPTRFR
Per the caller's code in R2, obtain a page frame -

DMKPTR, GETFREE
Obtain page frame via CORTABLE reference then exit to caller.

DMKPTRFE
Entry via CPEXBLOK, check page availability via flush list (DMKPTRFL), if none available steal a user's page.

DMKPTR, SELECT
The SELECT routine is entered to replenish the FREELIST from the flush list or user's pages that have not been referenced.

DMKPTEFT
Process pages to be returned by chaining them to the FREELIST. On page returns DEFER page requests are processed first.

DMKPTRLK
In locking a page in Real Storage (address in R2), add 1 to lock count; if previously locked, and exit to caller. If not previously locked, unchain the CORTABLE entry from the user's page list and set the lock count to 1.

DMKPTRUL
To unlock a locked page, reduce lock count by 1 and exit. If the lock count is now equal to zero, place CORTABLE entry on user's page list prior to exiting from routine.

Reading/Writing a DASD Page To/From Virtual Storage

Virtual Storage and Management - Non-EC Mode

DMKRPAGT
Entered via SVC call to read DASD page into storage.

DMKPGTPR
Release DASD space that was previously occupied by this virtual storage page.

DMKRPA, RESIDENT
Remove resident page frames from the user list.

DMKPTRFT
Place these page frames on the free list.

DMKRPA, STORDASD
Update the SWPTABLE with disk address in R0.

DMKPTRAN
Bring the page into storage.

DMKRPA, EXIT
Put real storage address of the virtual page is passed back to the caller in R2.

DMKRPAT
Entered via SVC call to write out a page to DASD storage.

DMKPTRAN
Locate the page to be moved and lock it.

DMKRPAPT
Store all registers in CPEXBLOK and flag CPEXR0 as a write request.

DMKPAGIO
Write the page.

DMKRPA, IORETN
Decrement page wait count. If zero results, take user out of page wait.

DMKPTRUL
Unlock the page frame. Return to caller.

Virtual Storage Management - EC Mode

DMKVATAB
Entry via BALR when an EC mode virtual machine needs a shadow table generation and update or purge operation.

DMKVATMD
Get storage to create shadow table, Flag VMBLOK to show shadow table existence.

DMKVATBC
Free shadow page, segment and copy segment, when user leaves EC mode or alters CR 0.

DMKVATSI
Entry via BALR to selectively invalidate a shadow page table entry.

DMKVAURN
Entry to perform third level to first level translations and third level translations to second level address translations. Use TRANS macro to access virtual segment and page tables to get the virtual page into real storage.

DMKVATLA
Using the TRANS macro to access the virtual segment and page tables, pass the resulting page and displacement to DMKPRVLG.

DMKVATPX
Invoked by DMKPRGIN when a paging exception is received for an EC mode virtual machine.

DMKVAT, SETUPEX
Perform set up operation and develop page table address.

DMKPTRAN
Get the page.

DMKVATPX
Update the shadow table.

DMKVATSX
Invoked by DMKPRGIN when a segment exception is received for an EC mode virtual machine.

DMKVAT, SETUPEX
Perform setup operation, then invalidate the shadow page table or if none exists, allocate a new shadow table and set it invalid.

DMKVATPF
Entered via DMKVATPG from DMKPRG to simulate pseudo page fault interrupts when a paging exception occurs with pseudo page fault interrupts enabled.

DMKPTRAN
Bring in the DASD page.

DMKPRGSM
Reflect program check `X'14'` to the user.

DMKVAT, PAGRES
When the page becomes resident in storage. Build the PGBLOK, set high order bit in the translation exception address field,

DMKDSPCH
Exit to dispatcher.

Allocation and Deallocation of DASD Space

DMKPGTPG
Entry to search and allocate a DASD page for paging.

DMKPGTSG
Entry to search and allocate one DASD page for spooling. Search appropriate RECBLOK chain for available DASD page. If none found, locate next available cylinder or FBA free space and construct a new RECBLOK, calculate address of the allocated DASD page and place it in R1. Return to caller.

DMKPGUPR/DMKPGUSD/DMKPGUSP
Entry to deallocate DASD page used for paging or spooling. Via ALOCBLOK locate the RECBLOK and reset appropriate bit in the RECBLOKs RECMAP and adjust the member of DASD pages in use. If all the pages on the DASD cylinder have been deallocated, deallocate the cylinder. Exit to caller.

DMKPGUSR
Entry to release a group of DASD pages no longer needed for spool file use. Per R1, find RECBLOK and dummy RECBLOKs and reset the RECMAP bits as specified. Free related RECBLOKS, if complete deallocation occurs.

DMKPGTCG
Entry for allocation of enough contiguous DASD spool space to record a 3704/3705 dump. Scan ALOCBLOK for enough contiguous available space to record the dump. When found, flag cylinder as allocated and build and chain the required RECBLOKs.

DMKPGTDT
Entry to allocate contiguous DASD space for a CP system dump.

DMKPGUDU
Entry to deallocate contiguous DASD space that was previously allocated for a CP system dump.

DMKPGUVG
DMKPGT contains an internal table, PAGETABL, in which the allocation of page frames for the CP paging VMBLOK is kept. The PAGETABL is scanned for a zero bit denoting the page frame is available. The page is marked allocated by setting the bit to one and the address of the page frame is returned to the caller in R1. If no page frames are available, a CPEXBLOK is built and queued to the deferred request chain.

DMKPUUVB
Entry to release a page of virtual storage. Check the chain of deferred requests. If there are none, reset the page bit in the PAGETBL to 0 and exit to the caller. Otherwise, give the page to the first requestor in the deferred chain and stack his CPEXBLOK for the dispatcher.

Shared Segment Storage Management

DMKATSCF
Entry via SVC from the command processor if an ADSTOP, TRACE, or STORE command is to alter a shared page. The virtual machine issuing the CP command will be unshared from the named system, that is, given a private copy.

DMKERMSG
The running virtual machine is informed of the share page violation.

DMKVMASH
Entered from DMPDSP or DMKPTR via BALR. The protected shared page tables are examined for hardware change bit being on. The resulting condition code is reflected to the caller.

DMKVMASW
Entered to switch the virtual machine from one set of page tables to the other.

Temporary Disk Storage Management

DMKTDKGT
Entry to allocate temporary disk space (T-disk). With R0 equal to the number of cylinders required and R1 equal to the device type, locate RDEVBLOK and related ALOCBLOK's ALOCMAP. If no allocation space is to be found, return to caller with 0 in R8. If allocation is successful, flag ALOCMAP, with X'AA' as allocated and put first cylinder address in R1 and RDEVBLOK pointer in R8 and return to caller.

Paging I/O Scheduler

DMKPAGIO
Entry to initiate page I/O activity. Uses preformatted IOBLOKs from the IOBSTACK for requests on count-key-data or fixed-block DASD, or DMKPAGEX for requests on extended count-key-data devices; fills in the CCWs with DASD operation code and values from CPEXBLOK, swap table, and core table. Chains the CPEXBLOK on the in-transit queue.

DMKPAG, GETRDEV
Find the Paging RDEVBLOK.

DMKPAG, FINDIOB
Search IOBLOKs seeking the same cylinder address. If found, chain the channel programs together with TICs.

DMKDSPCH
Exit to the dispatcher.

DMKPAG, QUEUEDIO
If no IOBLOKs with some cylinder address are found -

DMKIOSQR
Start the I/O operation.

DMKDSPCH
Exit to the dispatcher to await interrupt.

DMKPAH, UNTRANS
Upon interruption return, unchains the CPEXBLOK from the in-transit queue.

DMKSTKCP
Stack all deferred requests for execution.

DMKPAH, UNSTACK1
Returns IOBLOK to IOBSTACK or DMKPAGEX.

DMKDSPCH
Exit to dispatcher.

Release Virtual Storage Pages

DMKPGSSS
Entry to release partial virtual storage. Per R1 (address of first page to be released) and R2 (address of last page to be released) set partial entry flag.

DMKPGSPO
Entry to check for shared segments and decrement usage count. Store registers and flag full entry condition. Examine VMSHRSYS for shared segments. If so, decrement use count. On zero use count unchain the SHRTABLE from the active list.

DMKPGS, CKCLEAR
On NOCLEAR exit to caller. If not, store number of release pages in R8.

DMKPGS, PGOUT2
Locate page and swap tables for the segment to be released and index to the entry for the first page.

DMKPTRAN
Initiate paging, and when paging stops release the page frame.

DMKPGS, NEXTPAGE
8 value

DMKPGSPS
Exit to caller.

DMKPGSPS
Entry to release storage containing a named system passed by the caller. If register one is nonzero, search the page tables looking for a header equal to the named system. If found, release the swap and page tables and build new ones, if the address range still lies within the user's virtual storage size. If register one is zero, release and rebuild swap and segment tables for all segments above the normal virtual storage size that do not have SHRTABLE entries.

Free Storage Management

DMKFRTTR
Entry to assign storage to free storage management during CP initialization and whenever DMKPTRFR obtains a page from the dynamic paging area for use as free storage.

DMKFREE
Entry to obtain a block of storage, validate input doubleword request (R0).

DMKFRERC
Entry to obtain a block of free storage and return a condition code one if the request cannot be satisfied.

DMKFRE, CHEKSIZE
Checks the size of the request. For subpool size, see "DMKFRE, FREESUB" entry. For larger than 33 doublewords, see "DMKFRE, FREE01" entry.

DMKFREE, FREESUB
ON subpool size request, index into SUBTABLE. For correct size block found, remove block from chain and put the address of the block in R1. Return to caller.

DMKFRE, FREE02
For a subpool size that is not found, check the large storage chain. For an equal or larger block found, remove the requested block size from the larger chain, put the address in R1 and return to the caller.

DMKFRE, TRYSPLIT
For subpool and large storage chain that cannot honor request, start searching at 33 doubleword subpool for block requirement. When a block is found, split block (if necessary) and give caller address of his portion in R1 and chain the remainder to the appropriate subpool size. Return to caller.

DMKFRE, CLEARSAV
If no block can be found to honor user request, call -

DMKPTRFR
Fetch a page from the dynamic paging area. Chain it to the free storage chain. Processing then continues. See "DMKFRE, CHEKSIZE" entry.

DMKFRE, FREE01

For a large block request, search the large storage chain. For an equal or larger block found, remove block of request size from the chain, put the address in R1, and return to the caller. Otherwise, see "DMKFRE, CLEARSAV" entry.

DMKFRTRS

Entry to return all subpool blocks to the free storage chain per the SUBTABLE reference. As each subpool block is released, its address and length are placed in R1 and R2 respectively. Branch and link to FRET05 to return the block to the free storage chain (DMKFRELS). Repeat action through all subpools. Call DMKFRTSN to search the chain to return pages to the dynamic paging area. Return to caller.

DMKFRTSN

Entry to search the large storage chain for blocks needing to be returned to the dynamic paging area.

DMKFRET

Entry to return block to subpool or free storage chain. If the CP Assist Fret function cannot return the block to an appropriate subpool, transfer control to entry "DMKFRTT." Otherwise, attach the block to the appropriate subpool, update the pointer in SUBTABLE and return to the caller.

DMKFRTT

Entry to handle requests to return storage that cannot be handled by the CP Assist Fret instruction at DMKFRET.

DMKFRT, FRET01

Return a block of storage to free storage chain by merging into the chain storage addresses in an ascending order of sequence. Return to caller if the block is not the size of a page or more.

DMKFRT, FRET22

If the block includes a whole page frame or more and is in the dynamic paging area, branch and link to FRET22J, which calls DMKPTRFT to give the block back for paging. Call DMKFRTSN to search the large storage chain for other blocks needing to be returned to the dynamic paging area. Return to caller.

DMKFRTTE

Entry to return storage (known to have been obtained from the dynamic paging area) of subpool size or greater to the free storage chain.

CP Initialization and Termination Procedures

Loading the CP Nucleus on a Cold Machine

The value of CPID controls the actions of DMKCKP. For a cold machine, CPID is neither "CPCP" nor "WARM." In this case, DMKCKP simply loads DMKSAV and transfers control to DMKSAVRS.

IPL
The 24 byte IPL sequence containing the IPL PSW and a read CCW is loaded from the IPL device. The read CCW reads DMKCKP from the IPL device into location X'1000'. (The load point for a V=R system is the V=R size plus X'2000', or DMKSLC plus X'1000'.) The IPL PSW with an instruction address pointing to DMKCKPT is loaded.

DMKCKPT
Initial entry point to load the system.

DMKCKP, READREST
Load the rest of the checkpoint program (DMKCKD, DMKCKF, DMKCKH, DMKCKM, and DMKCKN).

DMKCKP, COLD
If CPID is not equal to "CPCP" or "WARM," then this is a COLD machine. Move "COLD" into CPID. Load both pages of DMKSAV into the storage location assigned by the loader during system generation. Go to DMKSAVRS.

DMKSAVRS
Load the nucleus up to DMKSAV. For processors with the Test Block facility, issue a Test Block to each 4K block of main storage before loading that page from the nucleus area. Go to DMKCPINT to initialize the system. See System Initialization for more details.

System Initialization

The value of CPID controls the actions of the system initialization routines. If CPID = "COLD," then this is a normal system IPL. If CPID = "WRM," then this is an automatic WARM start re-IPL following a system abend or SHUTDOWN REIPL.

DMKCPINT
Entry point to perform system initialization.

DMKCPI
Initialize the new PSWs, the control registers, the TOD clock comparator, the CPU timer, and the CPU id field of the PSA.

DMKCPI, INITREAL
Check if multiprocessing hardware is present.

DMKCPI, CHKCSS
Check if the channel set switching feature is installed.

DMKCPI, APCHKADR
For systems defined as AP during system generation, check if the other processor is available.

DMKCPI, CALLSTA
Call DMKSTANT.

DMKSTANT
Determine the real storage size, initialize the CORTABLE, allocate free storage and initialize system paging tables.

DMKCPI, CLEARCPA
If the ECPS microcode assist is not present or the wrong level, then replace each microcode instruction in the system with a NO-OP.

DMKCPI, OBTNSAVE
Obtain SAVEAREAs for use by DMKSVC, storage for use by DMKPTR in case the system needs to extend, and a work area for DMKIOS.

DMKCPI, GTXBLOOP
For systems defined as AP during system generation, if the other processor is available, then get storage for use by the switch macro and call DMKAPIPR.

DMKAPIPR
Called only if the other processor is available. Initialize the PSAs for both processors and start prefixing on both processors.

DMKCPI
Call DMKMNTIO.

DMKMNTIO
Entry point to mount I/O devices defined during system generation which are available.

DMKMNT, DOHIO
Issue a HIO to each I/O device defined in DMKRIO. If the device is not available, then go to MOUNTDVI to get the next device. Otherwise continue.

DMKMNT, RELEASE
Issue a RELEASE CCW to each DASD and TAPE which is available.

DMKMNT, READLABL
Read the label from each DASD and TAPE which is available.

DMKMNT, BUILDRDC
: Build RDCBLOKs for FBA devices which are available.

DMKMNT, ALLOCSIO
: Read the allocation record from DASDs which are available.

DMKMNT, CHKMSS
: Issue a SUSPEND CCW to MSS devices which are available.

DMKMNT, CKCPOWND
: If this is a CP owned DASD, then call DMKALOCP.

DMKALOCP
: Entry point to process CP owned DASD.

DMKALO, FBALOC
: Create an ALOCBLOK for CKD DASD.

DMKALO, FBA001
: Create an ALOFBLOK for FBA DASD.

DMKALO, FLAGRDEV
: Mark this device's RDEVBLOK as CP OWNED.

DMKALO, SAVEDRCT
: Compute the displacement into the system file table for the SYSOWN volume. Save the directory and override pointers. If this is the IPL volume, then save the address of this table entry as the location of the primary directory pointer.

DMKALO, ALOQUEUE
: DMKPGT contains the anchors of three ALOCBLOK/ALOFBLOK chains for each device type. Add the ALOCBLOK or ALOFBLOK to the temp space chain, fixed head paging chain, and moveable head paging chain as appropriate.

DMKMNT, DEVONLIN
: For each available device, mark the device and control unit as online.

DMKMNT, MOUNTDVI
: Repeat DOHIO through DEVONLIN for each I/O device defined in DMKRIO.

DMKMNT
: If the second processor is available in a multiprocessor configuration, then repeat DOHIO through MOUNTDVI for the second processor.

DMKMNT, SHIF3
: Recompute the RDEVBLOK indexes if necessary.

DMKCPI
Call DMKSEGCP.

DMKSEGCP
Entry point to set up CP's segment, page, core and swap tables.

DMKSEG, VBUFFOK
Call DMKBLDRT to build CP's segment and page tables.

DMKSEG, GETSEG
Fill in the entries in CP's swap and core tables.

DMKSEG, CALCUL
Calculate the number of fixed head and moveable head paging pages.

DMKCPI, SSMTEST
Check to see if the VM assist is available.

DMKCPI, S370EPGM
Check to see if the 370E facility is available.

DMKCPI
Call DMKOPERC.

DMKOPERC
Entry point to locate the operator's console.

DMKOPE
Locate the operator's console.

DMKOPE, ONLINE
Move the system id into the status area.

DMKOPE, WRITEINT
If CPID is "WRM," then this is an automatic WARM start following a system abend or SHUTDOWN REIPL. If CPID is "WRM" and the system is restarting after an ABEND, then display "VM/SP system restart due to system failure." If CPID is "WRM" and the system is restarting following a SHUTDOWN REIPL, then display "VM/SP system restart due to shutdown REIPL."

DMKOPE, INITWRIT
Display "VM/SP Release *x*, Service Level *xxxx*; created on *mm/dd/yy* at *hh:mm:ss*"

DMKOPE
Read in the first warm start record and determine if an auto-IPL may be performed.

DMKCPI
Call DMKTODIN.

DMKTODIN
: Entry point to initialize the time of day clock.

DMKTOD
: Display "It is now *hh:mm:ss time-zone day mm/dd/yy*"

DMKTOD, GETDATE
: If CPID is not "WRM," then allow the operator to change the date and time. If CPID is "WRM," then this is an automatic WARM start following a system abend or SHUTDOWN REIPL.

DMKTOD, NOTCHNG
: Schedule a TRQBLOK for midnight tonight. This TRQBLOK will cause the midnight message to be displayed by DMKSCHMD.

DMKTOD
: Schedule a TRQBLOK for 60 minutes from now. This TRQBLOK will cause storage subpools to be cleaned up by DMKTMRFR.

DMKCPI, APCHECK
: For systems defined as AP during system generation, if the other processor is available, then call DMKAPICK and DMKCLKCK.

DMKAPICK
: Entry point to test the other processor's clock, test for the virtual machine assist on the other processor, and complete initialization of the other processor.

DMKCLKCK
: Entry point to synchronize the time of day clocks.

DMKCPI, DRCTSET
: If the IPL volume contains directory pointer, try to load it. Otherwise, notify operator and try to load first valid directory found in SYSOWN order. Each time DMKUDRBV fails, try another backup directory from SYSOWN list. If no more SYSOWN volumes then Abend CPI002. If DMKUDRBV is successful, continue. If a valid override pointer is found on the same volume with the directory, call DMKUDROV to load it. Otherwise, notify the operator that the system defaults are used.

DMKCPI
: Call DMKCPJNT.

DMKCPJNT
: Entry point to continue CP initialization.

DMKCPJ, TLOOP
: Verify that the interval timer is running.

DMKCPJ, OWNDECK
: Verify the the warm start, checkpoint, and error recording volumes are mounted. If one of the volumes is missing, issue message DMKCPJ912W to operator. Then load disabled wait state PSW

(code 009). If an AUTOIPL is requested, bypass getting the operator's input and initiate the WARM START.

DMKCPJ, WARMTEST
If CPID is not "WRM," then display "START ((WARM|CKPT|FORCE|COLD) (DRAIN)) | (SHUTDOWN) :" and get the response. If CPID is "WRM," then this is an automatic WARM start following a system abend or SHUTDOWN REIPL.

DMKCPJ, PROEND
If response was "SHUTDOWN," then load a disabled wait state PSW (code 006).

DMKCPJ, AUTOWARM
If response was "WARM," "CKPT," "FORCE" or "COLD," then call DMKWRMST.

DMKWRMST
Entry point to handle WARM, CKPT, FORCE or COLD starts. See the separate discussion of each type of start for more detail.

DMKCPJ
Call DMKIDUMP.

DMKIDUMP
Entry point to locate DASD dump space, update allocation counts, mark the 3705s as unloaded, and clear T-disk space.

DMKIDU, DMPALOC
Scan the ALOCBLOKs searching for enough contiguous CKD DASD space to hold a system dump.

DMKIDU, MARKCYL
Fill in the dump SPOOL file SFBLOK for CKD DASD.

DMKIDU, RECALOC
Create RECBLOKs for the dump cylinders on CKD DASD. Go to QUEUEIO.

DMKIDU, FBA005
Scan the ALOFBLOKs searching for enough contiguous FBA DASD space to hold a system dump.

DMKIDU, DUMPOK
Fill in the dump SPOOL file SFBLOK for FBA DASD.

DMKIDU, RECBLD1
Create RECBLOKs for the dump pages on FBA DASD.

DMKIDU, QUEUEIO
Call DMKIOSQR to queue the I/O necessary to write out the symbol table (DMKSYM). Go to the dispatcher (DMKDSPCH) to wait for the I/O to complete.

DMKIDU, NEXTRECB
Update the allocation counts for FBA devices.

DMKIDU, NOTFBA
Update the allocation counts for CKD devices.

DMKIDU
Build index for page allocation priority.

DMKIDU
Rechain the ALOCBLOK/ALOFBLOK queues to match the SYSORD parameters specified at system generation time.

DMKIDU, NR3705
Mark the 3705 RDEVBLOKs as not ready.

DMKIDU, RELTDK
Stack CPEXBLOKs to clear T-disk space.

DMKCPJ
Call DMKOPELO to logon the system operator.

DMKCPJ, MAPMSG
Display:

```
DMKCPJ957I STORAGE SIZE = xxxxx K,
        NUCLEUS SIZE = xxx K,
        DYNAMIC PAGING SIZE = xxxxx K,
        TRACE TABLE SIZE = xxx K,
        FREE STORAGE SIZE = xxxx K,
        VIRTUAL=REAL SIZE = xxxxx K
```

DMKCPJ, COLDSPEC
If "DRAIN" was not specified, then call DMKCSORD to DRAIN (or STOP) 1) all system punches and printers for a COLD start, or 2) all punches and printers which are not drained for WARM, CKPT, or FORCE starts.

DMKCPJ, PPMAP
Call DMKHVDPP to initialize the program product bit map.

DMKCPJ
Call DMKSEGWR.

DMKSEGWR
Entry point to read in all pageable modules between DMKSAV and DMKCKP in order to force them to the backing device (that is, to create page image copies).

DMKCPJ
Call DMKIDUSF to get a valid SPOOL file id for the dump SPOOL file.

DMKCPJ, STARTSYS
Call DMKIOEFL to format the error recording area if necessary and then DMKCPJ initializes the machine check new PSW.

DMKCPJ, LOAD37X
Call DMKNLDR to load 370X programs into the enabled 370X devices.

DMKCPJ, CPJENAB
Call DMKCPVAE to re-enable terminal and graphics devices and call DMKNETAE to re-enable lines and stations.

DMKCPJ
Call DMKOPEAC to auto start MONITOR and auto log the AUTOLOG1 virtual machine.

DMKCPJ
Initialize IUCV control blocks.

DMKCPJ, CPJMIH
Create and schedule missing interrupt handler TRQBLOKS.

DMKCPI, CPIEND
Display "DMKCPI966I Initialization complete."

DMKCPI
Call DMKOPEDC to disconnect the operator if the operator was not logged on when the system ABENDED.

DMKCPI, INITDONE
Go to DMKDSPCH to begin dispatching virtual machines.

Warm Start Processing

DMKWRMST
Entry point to handle WARM, CKPT, FORCE or COLD starts. DMKWRMST is called by DMKCPJ during system initialization. For a WARM start, R2=02.

DMKWRM, EN3705
For 37xx devices create CKPBLOKs from the information saved in the terminal buffer.

DMKWRM, ENR3270
Enable binary synchronous lines by clearing NICBLOK offline flag (if appropriate).

DMKWRM, ACNTRT
Reconstruct the saved accounting ACNTBLOKs from the warm start data and chain them from the accounting card anchor (DMKRSPAC).

DMKWRM, WARMLOG
Retrieve the log message from the warm start area and save it in a buffer pointed to by DMKSYSLG.

DMKWRM, WARMSPL
Reconstruct the printer, punch, reader and delete SFBLOK chains from the data on the warm start area.

DMKWRM, WARMRECA
Update ALOCBLOKs and create corresponding RECBLOKs from the data on the warm start area.

DMKWRM, WARMHOLD
Reconstruct the SPOOL hold queue blocks (SHQBLOKs) from the data on the warm start area.

DMKWRM, ENDER2
Call DMKCKSIN.

DMKCKSIN
Entry point to initialize the checkpoint area. See "Checkpoint Area Initialization" on page 296 for more detail.

DMKWRM
Call DMKWRNWM

DMKWRNWM
Entry point to checkpoint SFBLOKs and SHQBLOKs present at initialization onto the checkpoint area.

DMKWRM, WARMCLR
Zero the first 8 bytes of the first record on the warm start area. Save the starting time for this system (STARTIME) in the first record and in DMKRSPCV. DMKCKP will compare these two values to ensure that the correct volume is mounted before checkpointing the system.

Checkpoint (CKPT) and FORCE Start Processing

DMKWRMST
Entry point to handle WARM, CKPT, FORCE or COLD starts. DMKWRMST is called by DMKCPJ during system initialization. For a CKPT start, R2=20. For a FORCE start, R2=40.

DMKWRM, ENDER1
Call DMKCKVWM.

DMKCKVWM
Entry point to handle CKPT or FORCE start.

DMKCKTMP
Determine the device type and the start and end of the checkpoint area. Calculate the number of checkpoint slots and based on the number of slots, allocate the checkpoint maps (maximum of 6) and

the SPOOL file id bit map. Set up other program constants in DMKRSP.

DMKCKV, CKVWM1
: Read the first/next page from the checkpoint area.

DMKCKV, CKVWM2
: Get the first/next checkpoint slot on this page.

DMKCKV, CKVWM2B
: For real device slots, restore the checkpointed RDEVBLOK data.

DMKCKV, CKVWM2G
: For SPOOL hold queue slots, reconstruct checkpointed SPOOL hold queue blocks (SHQBLOKs) from the checkpointed data. Chain the SHQBLOKs off the SHQBLOK chain anchor (DMKRSPHQ).

DMKCKV, CKVWM3
: For SPOOL file slots, reconstruct checkpointed SPOOL file blocks (SFBLOKs) from the checkpointed data. Chain the SFBLOKs off the printer, punch, or reader SPOOL file chains as appropriate.

DMKCKV, CKVWM5
: For *non-dump* SPOOL file slots, allocate the DASD pages used by each reconstructed SPOOL file by beginning with the first page and following the SPOOL page linkage pointers (SPLINK blocks), allocating each page, until the last page is reached.

DMKCKV, CKVWM6E
: For *dump* SPOOL file slots, allocate the DASD pages used by each reconstructed SPOOL file by beginning with the first page and sequentially allocating each page until the last page is reached. (Dump SPOOL file pages are allocated as contiguous space and are not SPLINKed together as are standard SPOOL file pages.)

DMKCKV, CKVM6D
: For SPOOL file slots, add the SPOOL file id for this SPOOL file to the SPOOL file id bit map.

DMKCKV, CKVM7
: If there are additional slots on this checkpoint page, then go to CKVWM2 to handle the next slot.

DMKCKV, CKVM8
: Write the checkpoint page back out to the checkpoint area in order to record changes made to the checkpoint page. Go to CKVWM1 to read in the next page.

DMKCKV, CKVM9
: Mark the slot following the last checkpoint slot as the physical end of the checkpoint slots. Set the next SPOOL file id (DMKRSPID) to one more than the largest id that was restored. Return to DMKWRM.

DMKWRM, WARMCLR
: Zero the first 8 bytes of the first record on the warm start area. Save the starting time for this system (STARTIME) in the first record and in DMKRSPCV. DMKCKP will compare these two values to ensure that the correct volume is mounted before checkpointing the system.

The following explains the difference between a CKPT start and a FORCE start:

DMKCKV, MSG915E
: Display "DMKCKV915E Permanent I/O error on checkpoint area" when an error occurs reading or writing a page from the checkpoint area. For a CKPT start, load a disabled wait state PSW (code 00E). For a FORCE start, if the error occurred while reading or writing the first page of the checkpoint area, then load a disabled wait state PSW (code 00E). For a FORCE start, if the error occurred while reading or writing any record other than the first page, then go to CKVWM1 to get the next checkpoint record.

DMKCKV, MSG916E
: Display "DMKCKV916E Error allocating spool file buffers" when an error occurs allocating one of the DASD pages making up a SPOOL file. For a CKPT start, load a disabled wait state PSW (code 00E). For a FORCE start, delete the corresponding SPOOL file, and continue with the next checkpoint slot.

COLD start processing

DMKWRMST
: Entry point to handle WARM, CKPT, FORCE or COLD starts. DMKWRMST is called by DMKCPJ during system initialization. For a COLD start, R2=01.

DMKWRM, ENDER2
: Call DMKCKSIN.

DMKCKSIN
: Entry point to initialize the checkpoint area. See "Checkpoint Area Initialization" on page 296 for more detail.

DMKWRM
: Call DMKWRNWM

DMKWRMWM
: Entry point to checkpoint SFBLOKs and SHQBLOKs present at initialization onto the checkpoint area.

DMKWRM, WARMCLR
: Zero the first 8 bytes of the first record on the warm start area. Save the starting time for this system (STARTIME) in the first record and in DMKRSPCV. DMKCKP will compare these two values to ensure that the correct volume is mounted before checkpointing the system.

Checkpoint Area Initialization

DMKCKSIN
Entry point to initialize the checkpoint area.

DMKCKS
Call DMKCKTMP.

DMKCKTMP
Determine the device type and the start and end of the checkpoint area. Calculate the number of checkpoint slots and based on the number of slots, allocate the checkpoint maps (maximum of 6) and the SPOOL file id bit map. Set up other program constants in DMKRSP.

DMKCKS, CLRENTRY
Set aside checkpoint slots for system dump SFBLOKs, and the system's printer, punch, and reader RDEVBLOKs. Mark the slot following these slots as the logical end of the checkpoint slot map.

DMKCKS, MAPINIT
Mark the first slot in each of the remaining checkpoint slot maps as the logical end of the map.

System Shutdown

The system can be shutdown by using the SHUTDOWN command.

DMKCPSSH .
Entry point to handle SHUTDOWN command.

DMKCPS, CHKPOWER
Check to see if the POWEROFF parameter was specified on the SHUTDOWN command.

DMKCPS, GOTMP
For AP/MP systems, halt work on the other processor.

DMKCPS, SIGPSTOP
If in single processor mode, then issue a SIGP STOP to the other processor.

DMKCPS, NOSPM
If monitor is active, then call DMKMNISH to stop it. Go to DMKDSPCH to wait for the necessary I/O to complete.

DMKCPS, CLOSECON
Call DMKVSPCO to close the caller's open console SPOOL file and the operator's open console SPOOL file.

DMKCPS, DASDCH
Locate and record statistical data for DASD, tapes and 3800 printers.

DMKCPS, NOSPM2

If SHUTDOWN without REIPL, then set CPID to "CPCP" to indicate a system shutdown. If SHUTDOWN with REIPL, then set CPID to "WARM" to indicate a system shutdown and automatic warm start. Go to DMKDMPRS to load DMKCKP.

DMKDMPRS

Entry point to re-IPL the system.

DMKDMP, RESTART

Get IPL device address.

DMKDMP, SIOIPL

Issue IPL CCW to load the IPL sequence. The 24 byte IPL sequence containing the IPL PSW and a read CCW is loaded from the IPL device. The read CCW reads DMKCKP from the IPL device into location X'1000'.

DMKDMP, B1

Load the IPL PSW read in by the IPL CCW. The IPL PSW has an instruction address pointing to DMKCKPT. DMKCKPT is the entry point to save the system warm start data. See Saving System Warm Start Data for more details.

Saving System Warm Start Data

DMKCKPT saves the system warm start data whenever the CPID is "CPCP" or "WARM." If the CPID is "CPCP," then DMKCKPT was invoked by the SHUTDOWN command (without the REIPL option) or following a dump to a tape or printer. (Or the system was STOPPED and then IPLed without clearing main storage.) In this case, a disabled wait state PSW (code 008) is loaded after the system warm start data is saved. If CPID is "WARM," then DMKCKPT was invoked following a dump to DASD or following the SHUTDOWN command (with the REIPL option). In this case the system will be reloaded and reinitialized after the system warm start data is saved.

DMKCKPT

Entry point to save the system warm start data and reload the system if indicated.

DMKCKP

Load the rest of the checkpoint program (DMKCKD, DMKCKF, DMKCKH, DMKCKM, and DMKCKN).

DMKCKP, GETLIST

Copy the list of pointers from DMKRSP into DMKCKF. This list contains addresses needed by the checkpoint program. The beginning of the list is pointed to by the field ARSPPR in the PSA. This list must be copied because if the nucleus is rebuilt and then the system is shutdown, DMKCKP would be using addresses resolved by the loader for the *new* nucleus to checkpoint the *old* system. These addresses would not necessarily match.

DMKCKP
: Call DMKCKDEV.

DMKCKDEV
: Entry point to Halt I/O during system shutdown.

DMKCKD
: Drain pending I/O interrupts.

DMKCKD, TRYSGICR
: For MP systems, halt prefixing on the other processor, and drain pending I/O interrupts on the other processor.

DMKCKD, GETCON
: Find a usable path to a console.

DMKCKD, CHINDEX
: Get the RCHBLOK, RCUBLOK and RDEVBLOK for the first/next device defined in DMKRIO.

DMKCKD, NETWORK
: For 37xx, if the system will be automatically restarting, then save the NCP name, real device address, number of NICBLOKs, and a bit map defining which terminals are enabled in the warm start terminal buffer.

DMKCKD, TEST3851
: For 3851s, issue a suspend CCW.

DMKCKD, TESTTERM
: For enabled terminals and graphics devices, save the device address in the warm start terminal buffer, and issue a HIO to the device.

DMKCKD, NONTERM
: For non-terminal devices, issue a HIO to the device.

DMKCKD, NEXTDEV
: Go to CHINDEX to handle the next system generated device.

DMKCKP, CALLCKF
: Call DMKCKFIL.

DMKCKFIL
: Entry point to checkpoint a warm system.

DMKCKF
: Call DMKCKHST to validate the first record of the warm start area. Compare the clock value saved on the warm start area with the clock value saved in DMKRSPCV. DMKWRM saved the same value in both places when this system was IPLed. This ensures that the system is shutting down with the same volume it IPLed.

DMKCKF, OPLOGOFF
: If CPID = "WARM," then the system will be auto restarting. Save the status of the operator (logged on/logged off) in the first record of the warm start area.

DMKCKF, TERMPUT
: If CPID = "WARM," then the system will be auto restarting. Write the warm start terminal buffer containing the addresses of the enabled terminals out to the warm start area.

DMKCKF, DLM1
: Write the first delimiter record.

DMKCKF, NXTLOOP
: Create and save device and user accounting cards for active virtual machines.

DMKCKF, ACTCHAIN
: Save the accounting records.

DMKCKF, DLM2
: Write the second delimiter record.

DMKCKF, LOGNXT
: Save the log message.

DMKCKF, DLM3
: Write the third delimiter record. The third delimiter record contains the date, time and day of the system log message.

DMKCKF, SAVESF
: Save closed printer SPOOL file SFBLOKs.

DMKCKF, DLM4
: Write the fourth delimiter record.

DMKCKF, RSPPU
: Save closed punch SPOOL file SFBLOKs.

DMKCKF, DLM5
: Write the fifth delimiter record.

DMKCKF, RSPRD
: Save closed reader SPOOL file SFBLOKs.

DMKCKF, CKPCPTRP
: Save the last record of open CPTRAP SPOOL files.

DMKCKF, CKPMONIT
: Save the last record of open monitor SPOOL files.

DMKCKF, DLM6
: Write the sixth delimiter record.

DMKCKF, SAVEDEL
Save SPOOL file SFBLOKs which are on the delete queue. These SPOOL files will be deleted after the system warm starts. They are saved in order to keep the saved allocation records valid.

DMKCKF, DLM7
Write the seventh delimiter record. The seventh delimiter record contains the SPOOL file id count.

DMKCKF, OWNLP
Save the allocation records.

DMKCKF, DLM8
Write the eighth delimiter record.

DMKCKF, NEXTSHQ
Save the SHQBLOKs.

DMKCKF, DLM9
Write the ninth delimiter record.

DMKCKF
Call DMKCKHWM

DMKCKHWM
Entry point to write a valid first record out to the warm start area. The first record contains the STARTIME value from DMKRSPCV and the version of VM/SP. If a 'SHUTDOWN POWEROFF' command was issued, turn on the auto-IPL bit on the first warm start record.

DMKCKF
Call DMKCKHWM to write a valid first record out to the warm start area. The first record contains the STARTIME value from DMKRSPCV and the version "VM/SP."

DMKCKP
Call DMKCKMSV.

DMKCKMSV
Entry point to save the virtual machines indicated in the system name table.

DMKCKM, VMSAVEON
If there was an active V=R user and the dump was directed to DASD (CPID = "WARM"), then restore pages 1 through 4.

DMKCKM, ASGO
Call DMKCKNRD to fix up the RDEVBLOKs of the CP owned DASD so that DMKCKNIO can use them. Read in the system name table (DMKSNT).

DMKCKM, ASLOOP1
: Call DMKCKNIO to save the pages specified for each virtual machine. DMKSNT specifies the name of the saved system, the volume serial number of the CP owned DASD that this system is to be saved on, the DASD cylinder and page to begin saving this system on, the userid of the virtual machine that is to be saved, and the pages of the virtual machine that are to be saved.

DMKCKM, ASTOD
: Save the date, the time, the name under which this system is being saved, the userid of this virtual machine, the VMPSW, the general, floating point, and extended control registers, and the storage keys for the saved pages.

DMKCKP
: If this is a system shutdown (CPID = "CPCP"), then set CPID to "SHUT" and load a disabled wait PSW (code 008). If it is a SHUTDOWN POWEROFF from a first-level CP system, issue a POWEROFF DIAGNOSE instruction and load a disabled wait state PSW (code 008).

DMKCKP, AUTOWARM
: Move "WRM" into CPID to indicate that the system is performing an automatic re-IPL.

DMKCKP, WARM
: Load both pages of DMKSAV into the storage location assigned by the loader during system generation. Go to DMKSAVRS.

DMKSAVRS
: Load the nucleus up to DMKSAV. For 3081 processors, issue a Test Block to each 4K block of main storage before loading that page from the nucleus area. Go to DMKCPINT to initialize the system. See CP Initialization for details.

Dumping the System and Automatic Re-IPL

DMKDMPDK
: Entry point to write a system storage dump to the dump device. Entry occurs via ABENDxxx condition or by pressing system console RESTART button. DMKDMP saves PSA values and determines if the dump is full of just CP portion.

DMKDMP, DMPMSG
: Format and issue abend message to operator.

DMKDMP, DMPDASD
: For dump to DASD, write out CP storage or all storage to selected DASD device.

DMKDMP, DSKEND
: For dump to DASD, place the sending record number and the system file number in the dump file SFBLOK.

DMKDMP, RECSRCH
For dump to DASD, chain dump file RECBLOKs to RDEVBLOK, and link dump file SFBLOK onto the system reader chain.

DMKDMP, CKSEND
For dump to DASD, set CPID to "WARM" so that DMKCKP will automatically re-IPL the system. Go to RESTART.

DMKDMP, DMPTAPE
For dump to tape, dump CP storage or all storage to the selected tape drive per specified tape parameters. Leave CPID equal to "CPCP." Go to RESTART.

DMKDMP, DMPPRT
For dump to a printer, dump CP storage or all storage to the selected printer. Leave CPID equal to "CPCP."

DMKDMP, RESTART
Get IPL device address.

DMKDMP, SIOIPL
Issue IPL CCW to load the IPL sequence. The 24 byte IPL sequence containing the IPL PSW and a read CCW is loaded from the IPL device. The read CCW reads DMKCKP from the IPL device into location x'1000'.

DMKDMP, B1
Load the IPL PSW read in by the IPL CCW. The IPL PSW has an instruction address pointing to DMKCKPT. DMKCKPT is the entry point to save the system warm start data. See Saving System Warm Start Data for more details.

Dynamic Checkpoint of Spool Files and Spool Devices

DMKCKSPL
Entry from any routine that adds, deletes, or changes the status of closed spool files. Lock the routine, or wait until it becomes unlocked. Bring the map page and spool fileid bit map page into storage and set up the device code of the system residence volume.

DMKCKS, LOOPSHQ
If the change is applicable to a SHQBLOK (hold queue block), make appropriate change on the checkpoint cylinder.

DMKCKS, CKSPL1
If the change is applicable to a SFBLOK, either add, change, or delete it on the checkpoint cylinder.

DMKCKS, CKSPL5
If the change affects a spooling device RDEVBLOK (for example, a START or DRAIN command issued), mark the change on the checkpoint cylinder.

DMKCKS, CKSEXIT
Check the routine. Unlock the page map and spool fileid bit map page and exit to caller.

Dump a Virtual Machine with VMDUMP Command

DMKVMDEP
Entry occurs via VMDUMP command. The command options are verified.

DMKPGUVG
Used to get virtual pages for use as spool files.

DMKRPAPT
Used to write out spool records.

DMKPGTSG
Used to obtain temporary space CCPD.

DMKPTRAN
Used to fix and lock storage pages.

DMKPTRUL
Used to unlock main storage pages.

DMKPGUVR
Used to release main storage pages.

Virtual Machine Initialization and Termination

Attaching a Virtual Machine to the System

DMKCNSIN
Entered via interruption from a console or terminal (not displays) device. If appropriate, determine and store device type in the RDEVBLOK. Write the VM/370 online message. Sets up to receive attention interruption.

DMKBLDVM
On attention interruption, build skeleton VMBLOK for LOGONxxx.

DMKCFMBK
Send read CCWs to the terminal for LOGON or DIAL response.

DMKTRMID
On response determine translate tables to be used.

DMKCFMBK
Validate command and transfer to DMKLOGON.

DMKLOGON
LOGON command execution.

DMKDIAL
Dial access linkage to multiaccess system.

DMKUDR
Via user directory access, validate user logon eligibility. On acceptance of eligibility, that is the successful completion of logon, build and allocate control blocks and linkages for the user's virtual machine.

IPL the Virtual Machine

DMKCFGIP
For the IPL of a named saved system, the name is verified and resources are checked for availability. Virtual storage is set up with the saved system via SWAPTABLE, SEGTABLE, SHRTABLE updates. For the IPL of device address, the IPL simulator is loaded in the user's storage. Before the IPL simulator is loaded into the user's storage, the contents of that storage is preserved and the area is then used for the IPL simulator.

DMKVMIPL
Read in 24 bytes from the CTCA, reader, DASD or tape unit into the user's virtual location zero. The CCW pointer is now set to the IPLCCW at virtual location X'8' and the IPL CCW string is executed.

DMKVMI, IPLDONE
Control comes here upon completion of the IPL CCW string. For IPL STOP, the virtual machine is placed in console function mode to allow change to nucleus name and apparent storage size before continuation.

DMKVMI, LOADNOW
IPL address is inserted in X'02' if BC mode, or X'BA', if EC mode. The user's CAW and registers are restored and control is given to the user by invoking diagnose X'10' to restore the page which the IPL simulator overlaid and load the current PSW at virtual location zero.

Virtual Machine Termination

DMKUSOLG
Entry is the result of user invoking LOGOFF. Set flags in VMBLOK indicating logout operation.

DMKUSOFL
Entry is the result of a Class A user issuing a FORCE command. Sets flags in the FORCE'd user's VMBLOK indicating logout operation.

DMKUSO, USO06
Retain line communication, if HOLD operand specified.

DMKUSO, USO08
Adjust return address to not run the user.

DMKUSOFF
Set VMBLOK flags. In the event of abnormal termination, save a virtual machine if it is enabled for VMSAVE.

DMKUSQFF
Continues logout processing. Called from DMKUSOFF.

DMKTRCND
Called to reset tracing.

DMKPENDA
Called to reset tracing.

DMKACOTM
Accounting called to compute the connect time for the LOGOFF message.

DMKQCNWT
Write the message to the user.

DMKSCHDL
Called to alter user dispatch status.

DMKCFPRR, DMKCSPO
Reset the virtual machine.

DMKMHCRE
Release HCBLOK for pending MSSF request.

DMKVMCAN
Release or return VMCBLOKs if VMCF is active.

DMKVATBC
Release shadow tables (if any).

DMKSCHRT
Dequeue clock comparator request (if any).

DMKBLDRL
Release segment tables, page and swap tables related to the user.

DMKUSO, USO94
Via DMKFRET return user VMBLOKs to free storage.

DMKUSO, USO93
For the system operator, clear and reinitialize the VMBLOK.

DMKFRET
Return all other virtual machine control blocks to free storage.

DMKACOFF
Punch an accounting card for the user.

DMKUSO, USO98
Free LOGOFF message area. Exit to do free storage maintenance. Exit to DMKCFM or DMKDSPCH.

DMKUSOFL
Entry is the result of the invoked FORCE command.

DMKSCNAU
Locate userid VMBLOK.

DMKUSOFL
Set VMKILL in VMBLOK, build CPEXBLOK and stack it for dispatcher.

DMKDSPCH
Upon CPEXBLOK execution, process as at LOGOFF entry DMKUSOFF.

DMKUSODS
Entry from an invoked CP DISCONN command. Set disconnected VMDISC in VMOSTAT.

DMKQCNWT
Send disconnect message to user.

DMKUSODS
Increment return address to DMKCFM by 4 to prevent a return read to the user's terminal. Clear VMTERM field to indicate the user terminal is disconnected.

DMKQCNWT
Send message to system operator informing him of user disconnect status. Exit to DMKCFM.

Console Function (CP Command) Processing

DMKCFMBK
Entry used when the ATTENTION key (or equivalent) is pressed once or twice (according to the VM or CP status) to allow the user to direct a line of input data for CP command processing. Set VMFCWAIT and VMCF bits in VMBLOK indicating wait state and console function mode.

DMKFREE
Builds an 18-doubleword CONBUF buffer for the read operation.

DMKQCORD
Read in the terminal input command line.

DMKSCNFD
Find the START and length of the command.

DMKCFCMD
Scans the CP command line, determines whether a requested CP console function is allowed for a user, and obtains the entry point of the command processor to handle the console function invoked on the command line.

DMKCFC, CMDCLC
Processes command with no subcommands.

DMKCFC, NEXTCMD
Processes command with subcommands.

See "Part 4: CP Diagnostic Aids" on page 405 for a list of all CP commands and the associated processing modules.

DMKQCORD
Read in the terminal input command line.

DMKCFMAT
On NULL data and ATTN key indication, post attention interrupt pending in VDEVBLOK, VCUBLOK and VCHBLOK. Return to run the virtual machine.

DMKCFMRQ
On receipt of CP commands ATTN or REQUEST, process the same as previous entry, DMKCFMAT.

DMKCFM
On receipt of * (asterisk) return to DMKCFMBK to set up another read. If console spooling is enabled, all console input and output including comments are spooled for printer output.

DMKCFMBE
On receipt of BEGIN, simulate the start button on the virtual machine (If optional address is supplied with BEGIN command the supplied address is substituted for the location counter address).

DMKCVTHB
Convert this address to binary notation.

DMKCFMSL
On receipt of the SLEEP command or SLEEP with time value (simulation of virtual machine stop button depression) the VMBLOKs VMSLEEP bit is set. The terminal console keyboard is now inactive until the user hits an ATTENTION key or the SLEEP command times out.

Dispatching and Scheduling

Fast Reflection for the Dispatched Virtual Machine

DMKDSPA
Entry for fast reflection activity. If the user is no longer runnable, or if the system is extending, the fast reflect path is not continued and processing continues at the main dispatcher entry point.

DMKDSP, UPVIRT
If the user is running virtual timers, update and test the user's virtual timers.

DMKDSPA1
If the user is still dispatchable, build the new RUNPSW from either IOOPSW or PROPSW and redispatch the virtual machine.

PSW Validation

DMKDSPB
Entry to dispatcher when the user's PSW has been external to DMKDSP.

DMKDSP, CKPSW
Verify the PSW change.

DMKDSP, CKPEND
Unstack any pending interrupts for the user (if enabled).

MAIN Dispatch Entry

DMKDSPCH
Normal dispatch entry after each interrupt handler has finished processing, and after each CPEXBLOK, I/O request and external interrupt has been serviced.

DMKDSP, RUNTIME
If CPSTATUS indicates return from running a user (CPRUN on), first ensure that supervisor time is being charged to RUNUSER. Check the user for time-slice end or queue-slice end, store the time remaining in the time-slice, and update processor problem state time. Also update virtual timers if running.

DMKDSP, WAITTIME
If CPSTATUS indicates return from wait (CPWAIT on), first ensure that supervisor time is being charged to the system. Determine the type of wait (I/O wait, page wait, or idle wait) and save the appropriate new wait time value.

DMKDSP, UNSTACK
For nonrunnable virtual machine, go to label CHKILL in DMKDSP.

DMKDSP, UNSTACK
For runnable user, check pending interruptions for the following:

- *DMKDSP, CKPEND*

  PER interruption (VMPERPND)
  If user PER is active, then reflect the PER event to the virtual machine. If CP PER is active, call DMKPERIL to handle the PER event.
  Pseudo page faults (VMPGPND)
  External interruptions (VMPXINT)

- *DMKDSP, UNSTIO*

  I/O interruptions (VMIOINT)

- *DMKDSP, STORECSW*

  I/O interruptions are reflected by swapping user PSWs and storing the unit address and status in low storage.

- *DMKDSP, CLEARVMX*

  Clear the pending bits in the VMBLOK.

DMKDSP, CKPSW
Validate the PSW.

- *DMKVATBC*

  For virtual machine leaving EC mode, clean up the shadow tables.

- *DMKVATMD*

  For virtual machine in BC mode and entering translate mode, initialize shadow tables.

DMKDSP, DSPERMSG
For PSW invalid, send error message to virtual machine, and place user in CP mode. If disconnected and invalid PSW, log off user.

DMKDSP, DISPATCH
Complete processing for current user. Call DMKSCHDL if necessary to alter user's dispatching priority.

Selecting the Next Unit of Work

DMKDSP, CKCPSTAK
Process a stacked request. First check the stack of IOBLOKs and TRQBLOKs. If system is not extending, unstack normally. Otherwise, only unstack paging or PCI IOBLOKs.

DMKDSP, WINDOW
Before examining the stack of CPEXBLOKs, open a window for interrupts if the system is not extending.

DMKDSP, CKCPREQ
Check the stack of CPEXBLOKs. If the system is extending, only unstack those blocks that will allow the extend to complete. If the system is not extending, unstack normally. If a CPEXBLOK for the other processor is encountered, give up the system lock and signal the other processor.

DMKDSP, CKUSERS
If no stacked requests can be unstacked, select a user for dispatching. If the system is locked for running users (such as during extend), load a wait state. Scan the run list for a dispatchable candidate. If none is found, load a wait state. If there is also a runnable user for the other processor, signal the other processor. If a runnable user is found, set up to dispatch this user.

Scheduling Users for Execution

DMKSCDL
Main entry to maintain queues of runnable and eligible users and to alter the user's dispatching status and, at queue-drop time, calculate his working set size and deadline priority. Also updates VMQBLOK statistical data.

DMKSCH, CKRSTAT
If the user is now not runnable, but was runnable before, mark the user as not runnable. If the user is in the eligible list, drop him from the list. If the user is in an idle wait state, drop him from the queue.

DMKSCH, CKRUN
If the user is now runnable, mark him as runnable. If the user was not in Q before, add him to the eligible list.

DMKSCH, CKWAITNG
Look through the eligible list for runnable users to add to active queues.

Page Migration

DMKPGM, DMKPGMEP
Invoked by the dispatcher if it has a CPEXBLOK stacked for page migration. It migrates pages from preferred backing store to non-preferred backing store. When page migration is completed, it checks to see whether the swap table migration flag is set. If it is, a CPEXBLOK is stacked in DMKSTKCP for swap table migration, entry point DMKSTRSM.

DMKPGM, DMKPGMUS
Invoked if page migration or swap table migration is invoked via the MIGRATE command. If command is valid, a CPEXBLOK is stacked with an entry point of DMKPGMX to migrate all users, or an entry point of RESETUS to migrate one user.

Swap Table Migration

DMKSTR, DMKSTRSM
Invoked if the dispatcher has a CPEXBLOK for swap table migration.

DMKSTR, DMKSTRAN
Invoked from DMKPTR if there is a segment exception.

System Performance Indicators

DMKSTP, DMKSTPX
At initialization time, control the generation of four system control constants. These control constants are then used in the calculation of the 28 system performance indicators, which in turn are used to produce six scheduling control fields. DMKSCH uses these control fields to calculate the deadline priority of the virtual machine at queue-drop time.

Other Scheduler Function

DMKSCHST
Set a clock comparator interrupt request.

DMKSCHRT
Reset a clock comparator interrupt request.

DMKSCHMD
Set up a request block for midnight date change.

DMKSCH80
Process a real interrupt timer request.

DMKSCHCP
Process a real CPU timer interrupt.

DMKSCHTI
Update system performance indicators.

Spooling Virtual Device to Real Device

Processing Virtual Output Files

DMKVSPEX
Entry from DMKVSI to initiate SIO on a spooling device that is available (not busy and no interruptions pending).

DMKVSTOP
If output device needs to be opened, call DMKSPLOV to build control blocks SFBLOK and VSPLCTL

DMKFREE
Get a work buffer for the CCWs and the data

DMKVSRGC
Get first CCW

DMKPGTVG
Obtain a virtual buffer; the address is stored in VSPVAGE.

DMKPGTSG
Obtain a DASD page; the address is stored in VSPDPAGE.

DMKVSP, PRINTER
Verify CCW opcode and set up initial CSW status.

DMKVSRMD
Get user's data in the work buffer.

DMKVSP, COMPRESS
Truncate all right-justified blanks.

DMKVSQ
If spooling space is available, move the CCW and data from the work buffer to the spool buffer; else call DMKPGTSG to get a new spool buffer and write out the full spool buffer.

DMKVSP, FLAGTEST
If the channel program ends (either all CCWs are processed or there is an error), go to LASTCCW; else, call DMKVSRGC to get the next CCW and go to DMKVSP, PRINTER.

DMKVSTCP
On console spooling, the following occurs:

1. Skip to channel 1 every 60 lines.
2. Write out the system console, spool file buffer every 16 lines.
3. Place the system console in a pseudo closed state for checkpoint recovery in the event of system failure.

DMKVSP, LASTCCW
When all CCWs are processed, post interruption pending to the VDEVBLOK, VDEVCSW and return control to the user.

DMKVSPPE
Close spooled printer/punch file via call to subroutine PRTEOF.

DMKVSP, PRTEOF
Page in the last page buffer if not resident. If the spool file is empty, purge it; otherwise, update pointers, number of buffers, and maximum record size and write last page buffer to DASD. If PURGE was specified, call DMKSPKDL to purge the file; else, close the file via DMKSPLCV and call DMKFRET to fret the VSPLCTL.

Closing Virtual Output Files

DMKVSUCO
Entry via CP CLOSE command.

DMKVSPST
If device busy, defer close operation by building CPEXBLOK, stack it and exit to dispatcher.

DMKVSPPE
On device not busy, write final buffer page to DASD storage.

DMKSPLCV
Queue closed virtual printer or punch spool file to the real spool output device, or transfer the file to another user's virtual reader. Also update the SFBLOK with number of copies printed/punched, distribution code, hold status, and file owner ID. If VSPXBLOK with TAG data exists for the spool device, copy the TAG data to the TAG record in the first spool file data buffer.

DMKSPL, TXTXFR
If a `spooled to` file, queue to the end of the reader file chain. Otherwise, chain the SFBLOK to the designated real spool printer or punch.

DMKCKSPL
Checkpoint the new spool file block.

DMKSPL, SETPEND
For a `spooled to` file find a virtual reader with the proper class and in the ready state with no active file, and no pending interrupts. Then build an IOBLOK with IOBIRA of DMKVIOIN.

DMKSTKIO
Stack the IOBLOK.

DMKSPL, SETPEND
Exit to DMKVSP.

DMKSPL, TSTHOLD
For not `spooled to` files and not in user or system hold, find printer or punch with the proper class. Then build an IOBLOK with IOBIRA of DMKRSPEX.

DMKSTKIO
Stack the IOBLOK.

DMKSPL, TSTHOLD
Exit to DMKVSP.

Processing Virtual Input Files

DMKVSWOR
Entry to open a spool input file. If VDEVSPL=0 the file needs to be opened. Build VSPLCTL block and a work buffer. Search the system reader file chain per PSA linkage ARSPRD for a file with appropriate user and class.

DMKVSP, SETFLAG
On file-found condition, place first DASD page address in VSPLCTL, VSPDPAGE. Obtain a virtual buffer and retain its address in the VSPLCTL block.

DMKVSP, READER
Check the CCWs for validity, move and expand the data back to its original size and the data is moved from the work buffer to user's virtual storage.

DMKVSP, RDRCOUNT
On EOF, set SFBEOF bit in SFBLOK and return to caller.

Closing Virtual Input Files

DMKVSUCR
For CLOSE operation requested via console command and the device is busy, initiate a delayed close by constructing and stacking the CPEXBLOK for the CLOSE.

DMKVSP, RDREOF
For normal end of file and VDEVSFLG indicates continuous read.

DMKVSP, OPENCONT
Locate the next file and continue reading.

DMKVSP, LASTFILE
For last file, post end status in RDEVBLOK.

DMKVSP, FILECLR
For HOLD status file (VDEVSFLG=VDEVHOLD), call DMKCKSPL.

DMKCKSPL
Checkpoints the file.

DMKVSP, FILECLR
Unchain the file (except hold files) from the reader queue and call DMKSPKDL.

DMKSPKDL
Delete the file.

DMKVSP, DVICECLR
To clear the device, call DMKRPAGT.

DMKRPAGT
Releases the storage page.

DMKPGUVR
Releases the virtual buffer.

DMKFRET
Releases storage for the work buffer and VSPLCTL block.

Spooling to the Real Printer/Punch Output Device

DMKRSPEX
Entry from the dispatcher when an IOBLOK is unstacked with an interrupt for spooling unit record device. IOBRADD points to the RDEVBLOK RDEVTYPC input or output class.

DMKRSP, RSPLOUT
If RDEVSPL indicates an available spool device (not active),

DMKFREE
Get storage for a work buffer and build a RSPLCTL block and link it to RDEVBLOK, RDEVSPL.

DMKRSP, PRNXTFIL
Search printer and punch SFBLOK chains for corresponding device, form, class, destination, and if the file is converted. On a found condition, unchain the block, put its address in RSPSFBLK. The FLASH name specified in the SPOOL command, if FLASH is specified, must match the flash overlay name for a 3800 printer.

DMKSEPSP
If called, provides separators for output pages or cards.

DMKCTSET
If the device is a 3800 printer, call this module to set it up.

DMKRSP, PROCESS1
If required, load character arrangement tables and graphic character modifications.

DMKIOSQR
Bring first spool data DASD page to the work buffer and convert CCW addresses to real device addresses.

DMKRSP, PRNXTPAG
Start the spool device.

DMKRSP, REPEAT
Repeat the process until done.

DMKCKSPL
Reprocess and reaccess the buffer, if multiple copies are specified.

DMKSPKDL
Checkpoint records the change to COPY count.

DMKSPKDL
Delete the file on completion (unless HOLD specified). If the device is a 3800 printer, check for delayed purge.

DMKRSP, PRNXTFIL
Locate the next spool file to process.

DMKRSP, PRTIDLE
Processing for the device is complete as there are no more SFBLOK, for this device or the device was drained.

DMKFRET
Release work area and completed IOBLOK storage.

DMKDSPCH
Exit to the dispatcher.

Spooling to the Real Input Device

DMKRSPEX
Entry from the dispatcher when an IOBLOK is unstacked with an interrupt for a spooling unit record device. IOBRADD points to the RDEVBLOK RDEVTYPC input or output class.

DMKRSTIN
Handles the card reader.

DMKSPLOR
Assume there is no active file being processed on the real input file reader. The spooling operator has issued the START command to the device to open the reader.

DMKSPL, BUILDCTL
Build RSPLCTL and SFBLOK.

DMKPGUVG
Get virtual buffer and place its address in RSPVPAGE.

DMKPGUSG
Get DASD buffer and place its address in SFBSTART and RSPDPAGE, linked together by pointers.

DMKIOSQR
Start the reader.

DMKDSPCH
Await the interruption.

DMKRST, RDERGETID
Check that the first card in the buffer is the userid header. If so, proceed.

DMKRST, RDRCARDS
Preload the buffer with CCWs.

DMKIOSQR
Issue the SIO (SIO's of 42 cards per buffer load).

DMKRST, RDRSIO
Write the buffer to the DASD slot. Repeat until EOF detected.

DMKSPLCR
Close the file on EOF. Queue the file on reader spool chains.

DMKCKSPL
Add the spool reader file block to the checkpoint cylinder data.

DMKSPL, RDRPEND
If the file owner is logged on, and his virtual reader is available, an IOBLOK is constructed with device end pending -

DMKSTKIO
Stacks it.

DMKRST, RDREXIT4
Release storage for virtual buffer, RSPLCTL and the SFBLOK. Return to DMKRSP.

DMKDSPCH
Exit to the dispatcher.

Spool File Deletion

DMKSPKDL
With R7 not equal to zero, place the specified SFBLOK on the delete chain anchored to DMKRSPDL.

DMKCKSPL
Delete the SFBLOK from checkpoint cylinder data.

DMKSPKDL
Assume the delete routine is not running, build a CPEXBLOK to call DMKSPKDR. Set the DELSW=X'80' (delete routine active).

DMKSTKCP
Stacks it and exits to caller.

DMKSPKDR
On unstacking the CPEXBLOK, if the SFBLOK is a system dump file, calls DMKDRDDD.

DMKDRDDD
Deallocates DASD buffers.

DMKSPK, NEXTSFB
For complete allocation chains of RECBLOKS, call DMKPGTSR.

DMKPGUSR
Deallocate DASD buffer and return to storage held by the dummy RECBLOKs.

DMKSPK, DELSTART
For incomplete allocation RECBLOK chains, deallocate by calling DMKPGTSD.

DMKPGUSD
Deallocates a page at a time via SFBSTART and the IOBLOK until the last page is reached.

DMKFRET
Delete the SFBLOK, then go to DMKSPL and NEXTSFB.

DMKSPK, NEXTSFB
If the delete queue is not empty, process the next SFBLOK an identical manner. Continue until all SFBLOK deletions are complete then call DMKFRET.

DMKFRET
Delete the IOBLOK.

DMKDSPCH
Exit to the dispatcher.

Recovery Management Support Operation

Establishing the Error Recording Base

DMKIOEFL
Entry from CP initialization module to set up pointers to VM/370 error recording cylinders.

DMKIOGF1
The STIDP instruction stores processor version and model in CPUID of PSA.

DMKIOG, ISSUEINS
Check attached channels. If standalone channel on the 165 or 168, the address of the logout routines is stored in the DMKCCH module.

DMKIOG, CHANGEID
Set up pointers for machine check and channel check record area and extended logout areas.

DMKIOG, IOGMCHIN
Obtain storage for machine check record, extended logout area, and CPEXBLOK. The MCHAREA is also initialized.

DMKIOG, PASTDAVE
Determine the 90%-full and 100%-full capacity of designated error recording cylinders and store the amount in DMKIOEMX and DMKIOENI respectively.

DMKIOG, FINDREC
Check first record of the error recording cylinders for proper format. If invalid, reformat. If valid but clear, store pointer value in PSA as the first available slot for error record. If valid but used, search for first unused slot and store its value in PSA.

DMKIOGFR
If on a 3031, 3032, or 3033 processor, read frames from the SRF (service record file) device, and write them to the beginning of the error recording cylinders with unique record types.

DMKIOG, CYLFULL
When error recording area is full, inform the operator, and continue.

DMKIOEFL
Turn off the recording in progress switch and exit to caller.

Process the Machine Check Interruption

DMKMCHIN
Entry via the machine check PSW upon detection of an unrecoverable and nonfatal processor or storage error. Disable soft machine recording; store logout area on the error recording cylinders. The system is enabled for hard machine checks by pointing the PSW to the termination routine.

DMKMCH, ENHARD
For the virtual machine, store status in the VMBLOK.

DMKMCH, MCHSYSIL
For system damage, timing facility, uncorrectable retry, multibit storage error, post system operator message, and flag the system as terminated. If the fault occurred in problem state, terminate the active virtual machine.

DMKMCH, SOFTSTG
For corrected ECC or processor retry, update soft error count and record the error and dispatch the virtual machine.

DMKMCH, MCHSKIP
For multibit storage error in problem mode, exercise storage location to clear up or flag as unavailable (permanent error).

DMKMCH, MCHCHANG
On an altered page condition, the virtual machine is reset, otherwise, the error is recorded and the virtual machine is redispatched.

DMKMCH, SPFTEST
Storage key failure. Exercise the 2K page key. If CP area and solid error condition process as DMKMCH, MCHSYSIL, intermittent, restore the key and go to the dispatcher. If key failure and in virtual machine area if permanent error, mark page as unavailable, terminate the user. If intermittent condition, refresh the key and dispatch the virtual machine.

DMKMCH, VIRTERM
On conditions that cause the termination or reset. The error is recorded, and both the user and the operator receive status messages. Per the termination flag, VMBLOK, the user is logged off and control returns to the dispatcher or is reset via DMKCFPRR.

DMKMCH, CHANTERM
For channel inoperative errors. DMKACR is called to attempt to recover the failing channel or channels.

DMKCFPRR
Virtual storage is released, the virtual machine is flagged dispatchable and placed in console function mode.

DMKMCH, TERM
On a hard machine check while handling a machine check, the machine check new PSW is loaded with a wait state PSW and the current PSW is enabled for hard machine checks.

DMKMCH, MCHTERM2
Locate the system or the user's VMBLOK.

DMKMCH, OPCOM
Call DMKMCTPT if system is running in attached processor or multiprocessor mode.

DMKMCH, MCHWAIT
Load disabled wait state for uniprocessor system.

DMKMCTPT
Complete processor termination for attached processor or multiprocessor system. If the error is on the attached processor and it is in problem state, signal for automatic processor recovery and stop the attached processor. If the processor is executing in multiprocessor mode and the error occurred while the processor was in problem state, signal for automatic processor recovery and stop the failing processor.

DMKMCT, SWITCH
Make sure processing is on an I/O processor and set up the appropriate wait state code.

DMKMCT, OPCOM
Issue a message to the operator and load a disabled wait state for the attached processor or multiprocessor system.

DMKMCTPR
Perform automatic processor recovery function. Allow system to convert to uniprocessor mode by calling DMKCPPUP.

DMKMCT, PREXIT
Terminate the virtual machine if it is in control. Reset the main processor timer. Clear all lock words and return to the dispatcher.

Process the Channel Check Interruption

DMKCCHIS
Entry via DMKIOS via CSW channel error.

DMKFREE
Obtain storage and build a CCHREC block and if IOBLOK and RDEVBLOK exist, build an IOERBLOK.

DMKCCH, CCHIOERL
Store the CCHREC address, its length, and the CSW in the IOERBLOK.

DMKCCH, CCHDEPND
Call appropriate channel error analysis module. Analyze channel logout data for validity.

DMKCCH, SCNEND
Record the error on the error recording cylinder, if appropriate.

DMKCCH, CPTERM
Terminate CP if the PSA's terminate flag is set.

DMKCCH, CHANTERM
Attempt recovery from an I/O interface inoperative or a reset channel condition.

DMKCCH, CCHWAIT
Set up X'0F' wait state code and call DMKMCHST to terminate the system.

DMKMCHST
If the system is running in attached processor or multiprocessor mode, call DMKMCTST.

DMKMCH, CALLOPR
Issue an error message to the operator.

DMKMCH, MACHWAIT
Load a disabled wait state for a uniprocessor system.

DMKMCTST
Make sure system is running on an I/O processor; load disabled wait state.

DMKMCT, CALLOPR
Issue an error message to the operator.

DMKMCT, MFAWAIT
Load a disabled wait state for attached processor or multiprocessor system.

DMKCCH, SCNEND
Unless termination is established, return to DMKIOS for recovery.

Recording the Errors of the Virtual User Via SVC 76

DMKVERD
Entry via DMSPSA as a result of SVC 76 detection. Check parameters passed in R0 and R1.

DMKFREE
Obtain storage for a record buffer for the user error record.

DMKVER, BUFFUL
Using valid record type (from the buffer) branch to an appropriate routine to format that particular record type.

DMKVER, VER30
Using RDEVBLOK, VDEVBLOK and VMBLOK, convert virtual data to real values and place in record.

DMKIOERV
Record the error.

DMKDSPCH
Exit to dispatcher.

User Directory Routines

DMKUDRFU
Entry after CP detected LOGON command. DMKSYSPL points to the directory. Determine length of userid, if valid call DMKLOCKQ.

DMKLOCKQ
Lock the directory in storage.

DMKUDR, NXTPAGE
Bring in each directory page and return each page (and clear the buffer) until a UDIRBLOK match occurs or directory's last page is detected.

DMKUDR, FINDUSER
On userid found, move UDIRBLOK to caller's area.

DMKLOCKQ
Unlock the directory in storage.

DMKUDR, EXITCC0
Return to caller.

DMKUDRFD
Entry from calling routine to find the addressed (cuu) device UDEVBLOK in users directory and move it to the caller. Via UMACBLOK locate the UDEVBLOKs.

DMKUDR, FINDDEV
Check to see if the user device address is the same as in the UDEVBLOK. Search the chain until match or end of chain occurs.

DMKUDR, DEVFOUND
For found condition, post condition code zero in user's VMPSW.

DMKUDRRD
Entry from calling routine to read the UDEVBLOK addressed into the caller's buffer. Using the DASD and the user displacement from the UMACBLOK, bring in the buffer page to storage. Determine if the

virtual directory page address (UDBFVADD) exists in the user directory buffer blocks. If not call-

DMKPGUVG
and get a virtual page.

DMKRPAGT
For DASD address does not match the UMACBLOK, point to the DASD page and bring in the virtual buffer page. Move UDEVBLOK into callers area and set CC = 0 in VMPSW. Return to caller.

DMKUDRRV
Entry to return a virtual page used as a buffer. Determine if UDBFBLOK contains a virtual buffer page pointer (UDBFVADD). If not, exit-with CC = 1 set in the VMPSW. If a buffer exists, check to see if it is resident; if it is, clear it to zeros.

DMKPAGT
Return the real page to the system.

DMKRGTVR
Return the virtual page to the system.

DMKUDRRV
Set CC = 0 and return to caller.

DMKUDRBV
Entry from DMKDIRCT or DMKCPINT to build page buffers for each UDIRBLOK.

DMKFREE
Get storage for the virtual buffer page list.

DMKUDR, GETVPAGE
Call DMKPGTVG and DMKRPAGT to get the virtual and real buffer. Save the virtual buffer address in the page list.

DMKUDR, FRETLIST
Encountered I/O error, free the virtual buffer page list, post fatal message, set CC = 3 and return to caller.

DMKUDR, ENDLIST
Swap the new virtual buffer page list with the old list. Anchor the new list to DMKSYSPL.

DMKUDR, FRETLIST
If there was a previous buffer page list, free it. Save the start of the user directory pointer in DMKSYSUD, and return to caller with a CC = 0 in the VMPSW.

Save the 3704/3705 Control Program Image Process

DMKSNCP
Entry from DMKHVC and DIAGNOSE code 50. Per the system VMBLOK, locate the DMKRNTBL. The CCPARM virtual address is contained in R1 of the DIAGNOSE instruction.

DMKSNC, NAMECHK
Match via search CCPARM; CCPNAME with DMKRNTBL entries.

DMKSNC, SIZECHK
Verify DASD space requirements for 3704/3705 control program and resource data. The volume required to save (NCPVOL) as indicated in the NCPTBL entry must be available and mounted on the system, on a CP owned and supported paging device.

DMKSNC, SVRESDAT
Save resource data on the NCPVOL device. CCPARM supplies the starting address and size parameters for this write operation.

DMKSNC, SVNCPIM
Save 3704/3705 control program image on NCPVOL device. CCPARM also provides the parameters for this similar operation.

DMKSNC, SAVEFINI
Store CC=0 on no errors and return to caller.

Inter-Virtual Machine Communication

DMKVMCFC
Entry from DMKHVC and the DIAGNOSE instruction code X'68'. Builds a VMCBLOK and initializes it with data from the user's parameter list, VMCPARM. The virtual address of VMCPARM is contained in bits 8-11 (rx) of the DIAGNOSE instruction.

DMKVMC, VMCFTBL
Branch table to pass control to the appropriate subroutine based on the subfunction code in VMCPARM.

Subfunction Code	**Subroutines**
X'0000'	VMCAUTH
X'0001'	VMCUAUTH
X'0002'	VMCSEND
X'0003'	VMCSENDR
X'0004'	VMCSENDX
X'0005'	VMCRECV
X'0006'	VMCCNCL
X'0007'	VMCREPLY
X'0008'	VMCQIES
X'0009'	VMCRESUM

X'000A'	VMCIDENT

DMKVMC, VMCWAKUP
Notifies a virtual machine of a pending VMCF communication by posting a special external interrupt X'4001' unless:

- There is already a special external interrupt posted.
- The virtual machine is running disabled for VMCF interrupts (PSW bit 7 and CR0 bit 31).

DMKVMC, VMCXFER
Transfers data from one virtual storage to another virtual storage. Errors occurring during data transfer are reflected to originating virtual machine via the data transfer return code in the final response interrupt message header.

DMKVMCEX
Called from DMKDSP to reflect an external interrupt message header to a virtual machine. If the VMCF subfunction is a SENDX, the SOURCE data is moved into the external interrupt buffer immediately following the message header.

DMKVMCUA
Called by DMKCFP when a virtual machine is logged off or reset. Uses the VMCUAUTH subroutine (subfunction code X'0001') to dispose of existing VMCBLOKS before turning off virtual machine communication.

Inter-User Communications Vehicle

DMKIUAEP
Entry is a GOTO from DMKPRV when a B2F0 instruction is encountered in a virtual machine. An IUSAVE block is built to contain the information needed to process the request.

DMKIUACP
Entry is via a CALL from a CP module requesting an IUCV function on behalf of the system. An IUSAVE block is built to process the request.

DMKIUACU
Entry is via a CALL from a CP module requesting an IUCV function on behalf of a virtual machine. An IUSAVE block is built to process the request.

DMKIUAPD
The path description entry is located in the caller's path description segment.

DMKIUAPL
Validates a communication path for a system path. CP paths are enqueued or dequeued as requested.

DMKIUAQU
Entry to queue a MSGBLOK on an IUCV message queue. The block is queued in priority, then non-priority FIFO order. If the purge bit is on in the MSGBLOK, the MSGBLOK is not queued and DMKFRET is called to release the MSGBLOK space.

DMKIUA, CKIDLE
Resets the IUCV wait conditions to allow an IUCV external interrupt to be reflected.

DMKIUA, BTABLE
Branch table used to determine the module and subroutine to handle the IUCV request.

DMKIUBRK
Entry is via DMKDSP to determine if there are any IUCV interrupts to be reflected to the virtual machine. The interrupts are reflected in the following sequence: IUCV control interrupts, priority replies, non-priority replies, priority messages, and then non-priority messages.

DMKIUBTB
Table containing the CP system service entry points for connects, messages, severs, quiesces, and resumes.

DMKIUCEP
General entry for this IUCV module.

DMKIUC, ACCEPT
Handle IUCV request to complete a communications path.

DMKIUC, CONNECT
Handle IUCV request to establish a communications path.

DMKIUEEP
General entry point for this IUCV module.

DMKIUERC
Perform the copying of data from the source to the target virtual machine for APPC/VM RECEIVEs.

DMKIUE, RECEIVE
Handle IUCV request to receive the message and cause the actual data transfer. If the message did not require a response, the MSGBLOK is moved to the source's REPLY queue; otherwise, the MSGBLOK is moved to the target's RECEIVE queue.

DMKIUGEP
General entry point for this IUCV module.

DMKIUGGP
This routine is used to locate an available path description entry to be used in establishing a connection.

DMKIUG, PURGE
Handle the IUCV request of the source to cancel a message. The MSGBLOK is located and marked as purged.

DMKIUG, REJECT
Handle the IUCV request of the target to refuse a message. The MSGBLOK is moved to the source's REPLY queue and marked as rejected.

DMKIUJEP
General entry point for this IUCV module.

DMKIUJ, SEVER
Handle the IUCV request to sever an IUCV or APPC/VM path.

DMKIULEP
General entry point for the IUCV module.

DMKIULRP
Perform the copying of data from the source virtual machine to the target virtual machine for APPC/VM SENDS when there is a predefined receive or answer area defined by the target.

DMKIUL, REPLY
Handle the IUCV request to answer a message and cause the actual data transfer of the reply. The MSGBLOK is moved to the source's REPLY queue.

DMKIUL, SEVER
Handle IUCV request to halt communications over a particular path.

DMKIUL, TESTCMPL
Handle IUCV request to test completion of a previously sent message. If the message has completed, the MSGBLOK is moved to the source's REPLY queue.

DMKIUNEP
General entry point for this IUCV module.

DMKIUNIN
Stack an external interrupt of type X'4000' to indicate an IUCV event.

DMKIUN, DESCRIBE
Handle IUCV request to describe a message in the invoker's SEND queue.

DMKIUN, SEND
Handle IUCV request to send a message across a communications path. A MSGBLOK is built and queued on the target's SEND queue.

DMKIUN, SETCMASK
Handle the five types of IUCV control interrupts. These control interrupts are: Connection Pending, Connection Complete, Path Severed, Path Quiesced, and Path Resumed. Only those bits set are enabled.

DMKIUN, SETMASK
Handle the IUCV request to change the external interrupt submask. Only those bits set to one are enabled and only those type IUCV interrupts are reflected.

DMKIUN, TESTMSG
Handle IUCV request to test a message. If messages are queued, the condition code is set to one. If replies are queued, the condition code is set to two. If both are queued, the condition code is set to three.

DMKIUPEP
General entry point for this IUCV module.

DMKIUP, DCLBFR
Handle IUCV request to declare a buffer to contain the external interrupt data.

DMKIUP, QUERY
Handle IUCV request to determine the length of an IUCV external interrupt buffer and the maximum number of connections allowed for this virtual machine.

DMKIUP, QUIESCE
Handle IUCV request to prevent incoming messages on the established communications path.

DMKIUP, RESUME
Handle IUCV request to allow incoming messages on a previously quiesced communications path.

DMKIUP, RTRVBFR
Handle IUCV request to retrieve a buffer and terminate IUCV communications.

DMKIUSEP
General entry for this IUCV module.

DMKIUSET
To set the state of a APPC/VM path when a function completes.

DMKIUS, RECEIVE
Handle APPC/VM request to receive a message or signal and to cause the actual data transfer.

DMKIUS, SEND
Handle APPC/VM request to send a message or signal across a communication path.

Console Communication Services

The Console Communications Services (CCS) function of CP links with the VTAM SNA Console Support (VSCS) to provide support for SNA terminals. The processing paths described below are the major paths used by CCS.

Unsolicited Interrupt

DMKVCPIL
Entered via call from IUCV (DMKIUN). Register 1 contains IUCV external interrupt buffer. For this path DMKVCPIL is the controlling module that calls all other modules.

DMKIUACP
Does IUCV receive.

DMKVCVKS
Locates SNA control blocks.

DMKVCPIL
Decodes function code (WEBFUN) to determine operation (Attention interrupt with no data).

DMKCFMAT
Posts an attention to the virtual machine.

DMKVCPIL
Stacks a CPEXBLOK to dispatch the virtual machine.

DMKVCRNR
If a batched write is pending, send the write.

DMKVCPIL
Exits to caller.

If data was received with Attention interrupt, DMKVCPIL determines the environment.

DMKFREE
If VM environment, obtains a buffer.

DMKVCPIL
Moves the WEBLOK's WEBDATA field to the buffer.

DMKFREE
Obtains space for a dummy CONTASK for editing. Edits the data. Releases storage for CONTASK. Determines if #CP is in the data.

If #CP not in data, DMKVCPIL chains the buffer address off the virtual machine's VCONRBUF.

DMKCFMAT
Posts attention to virtual machine.

DMKVCPIL
Stacks a CPEXBLOK to dispatch the virtual machine.

DMKVCVEB
Builds a WEBLOK.

DMKVCVLY
Issues an IUCV reply to indicate input data accepted.

DMKFRET
Releases the buffer storage.

DMKVCVNR
If a pending batched write exists, sends the write.

DMKVCPIL
Exits to caller.

If #CP was in data, DMKVCPIL calls:

DMKCFMEN
Passes input to CP.

DMKVCVEB
Builds a WEBLOK.

DMKVCVLY
Issues an IUCV reply to indicate input data accepted.

DMKFRET
Releases the buffer storage.

DMKVCVNR
If a pending batched write exists, sends the write.

DMKVCPIL
Exits to caller.

If not VM environment, DMKVCPIL calls:

DMKFREE
Obtains a buffer.

DMKVCPIL
Moves the information from WEBLOK's WEBDATA field to the buffer.

DMKFREE
Obtains space for a dummy CONTASK for editing.

DMKSCNED
Edits the data.

DMKFRET
Releases the CONTASK storage.

DMKCFMEN
Passes input to CP.

DMKVCVEB
Builds a WEBLOK.

DMKVCVLY
Issues an IUCV reply to indicate input data accepted.

DMKCFRET
Release the buffer storage.

DMKVCVNR
If a pending batched write exists, sends the write.

DMKVCPIL
Exits to caller.

Initiate A Read

DMKVCRRD
Called via SVC from DMKQCORD. For this path DMKVCRRD is the controlling module and calls all other modules.

DMKSCHRT
If a timer has been set for the user, resets it.

DMKVCVIX
If a batched write exists, sends the write.

DMKVCVCE
Writes a Trace Element.

DMKFREE
Gets storage for a WEIBLOK.

DMKVCVEB
Builds a WEBLOK.

DMVCVIX
Does a two-way send of the Read to the virtual service machine.

DMKVCVCE
Writes a Trace Element.

DMKVCRRD
Exits to Dispatcher.

Return From A 'READ'

DMKVCRMT
Invoked from IUCV (DMKIUA). At entry all registers are set up by IUCV. For this path DMKVCRMT is the controlling module and calls all other modules.

DMKVCVCE
Writes a Trace Element.

DMKVCRMT
Decodes the WEBLOK for the function, mode, and logical aid (Enter key). Moves the data from WEBBLOK's WEBDATA field to a buffer obtained via a call to DMKFREE. Edits the data for line end characters and tab characters. Moves the data to the user buffer pointed to by the CONTASK.

DMKSCNED
Edits as directed by the CONPARM.

DMKVCRMT
Stores a return address in the save area pointed to by CONRETN.

DMKVCVER
Sets a redisplay timer for the virtual service machine.

DMKVCVIN
Frees the WEIBLOK, IXBLOK, and CONTASK built when the Read was initiated.

DMKVCVND
If any Writes are chained to the SNARBLOK, sends then to the VSM.

DMKVCVND
If any Reads are chained to the SNARBLOK, sends them.

DMKVCRMT
Exits to Dispatcher via GOTO.

Initiate A WRITE

DMKVCSWT
Entered via branch from DMKQCN. For this path DMKVCSWT is the controlling module and calls all other modules. Determines type of Write by examining the CONTASK:

If Full Screen Write, or Diagnose Write, and if a batched Write is pending, calls:

DMKVCVIX
Sends the batch write to the virtual service machine.

DMKVCVLD
Obtains a WEIBLOK.

DMKVCVEB
Obtains a WEBLOK.

DMKVCSWT
Moves the output line from the CONTASK to the WEBLOK. If Console Mode, sets on the batching bits in the SNARBLOK.

DMKVCVER
Sets a timer.

DMKVCSWT
If WEBLOK is to be send immediately, that is if CONPARM = priority, or highlight, or alarm, or the pace counter has reached zero, calls:

DMKVCVIX
Sends the WEBLOK.

DMKVCVCE
Writes a Trace Element.

DMKVCSWT
If a response is expected to the Write, exits to Dispatcher via GOTO. If a response is not expected, sets the return code in the save area pointed to by CONRETN.

DMKQCOET
Release the CONTASK.

DMKVCSWT
Exits to Dispatcher via GOTO.

Part 3: CP Directory

This part contains the CP Module Entry Point Directory.

Chapter 8. CP Module Entry Point Directory

Module Name	Entry Points	Attributes, Function
DMKACO		Pageable, executable.
	DMKACOCL	Closes the current accounting spool file, executable.
	DMKACODS	Creates an account card buffer for a VMBLOK (disconnected user), executable.
	DMKACODV	Builds an account card buffer for a VDEVBLOK, executable.
	DMKACOFF	Creates an account card buffer for a VMBLOK (logged off user), executable.
	DMKACON	Provides additional accounting function at logon time (for installation use), executable.
	DMKACOQU	Collects the accounting records on the system accounting chain (DMKRSPAC) and puts them into spool file format, executable.
	DMKACOSA	Creates an accounting buffer for a VSCS logical unit user, executable.
	DMKACOTM	Creates a connect and usage time message for a user, executable.
DMKACR		Resident, executable.
	DMKACRCO	Marks all paths through specified channels offline, and checks that there are still online paths to all system-owned devices, executable.
	DMKACRCT	Terminates one or more failing channels, executable.
	DMKACRC3	Send an offline message to the operator, executable.
	DMKACRIO	Entry from DMKIOS by stacking a CPEXBLOK, executable.
DMKACS		Resident, executable.
	DMKACSCV	Recovers from loss of one or more channels or from the loss of a processor and its channels, executable.
	DMKACSRF	Internal ACR/ACS channel recovery flag, non-executable.
DMKALG		Pageable, executable.
	DMKALGON	Handles the AUTOLOG command, executable.
DMKALO		Pageable, executable.
	DMKALOCP	Special processing required for mounting CP owned volumes during system initialization, executable.

Module Name	Entry Points	Attributes, Function
DMKAPI		Pageable, executable. This module is entered from DMKCPI only if in attached processor or multiprocessor mode. It is also entered from DMKCPU as part of the vary online processor function.
	DMKAPIAP	Initializes the control registers for the second processor, executable.
	DMKAPICK	Tests for virtual machine assist and 370E on non-IPLed processor. Also tests if its interval timer is running, executable.
	DMKAPIPR	Initializes the PSAs for each processor, executable.
DMKAPS		Pageable, executable.
	DMSAPSCN	Handles pending CONNECTs from a virtual machine to the *SPL System Service, executable.
	DMKAPSIL	Handles pending messages representing spool requests for the *SPL System Service, executable.
	DMKAPSSV	Handles pending SEVERs from a virtual machine to the *SPL System Service, executable.
DMKAPT		Pageable, executable.
	DMKAPTRX	Handles pending messages from a virtual machine requesting the *SPL System Service read the XABs (External Attribute Buffers) for the spool file id specified into the virtual machine's data area, executable.
	DMSAPTSF	Handles pending messages from a virtual machine requesting the *SPL System Service read the SFBLOK for the spool file id specified into the virtual machine's data area, executable.
	DMKAPTSP	Handles pending messages from a virtual machine requesting the *SPL System Service read SPLINK(s) into the virtual machine's data area, executable.
DMKAPU		Pageable, executable.
	DMKAPUSE	Handles pending messages from a virtual machine requesting the *SPL System Service select a spool file from the system queue for the virtual machine to process, executable.
DMKAPV		Pageable, executable.
	DMKAPVCL	Handles pending messages from a virtual machine requesting the *SPL System Service to either close and purge, close and requeue, or change the copy count of the spool file specified, executable.
DMKAPW		Pageable, executable.
	DMKAPWPG	Removes a *SPL System Service lsplctl (logical spool control block) from the logical printer for a virtual machine, executable.
DMKAPX		Pageable, executable.
	DMKAPXMG	Sends messages from a virtual machine using the *SPL System Service, executable.
DMKAPY		Pageable, executable.
	DMKAPYSD	Routes commands and messages from the *SPL System Service to the virtual machine, executable.

Module Name	Entry Points	Attributes, Function
DMKAPZ		Pageable, executable.
	DMKAPZNO	Notifies all idle virtual machines which are using the *SPL System Service when a spool file becomes available, executable.
DMKATS		Pageable, executable.
	DMKATSCF	Notifies the virtual machine that the command has replaced the shared system with a private copy of that shared system. The user continues to run without the shared copy of the named system. Called by the command processors via an SVC if the command execution is to change a shared page, executable.
DMKBIO		Pageable, executable.
	DMKBIOCN	Handles pending CONNECTs from a virtual machine to the DASD Block I/O System Service, executable.
	DMKBIOIL	Handles pending messages representing I/O requests for DASD Block I/O, executable.
	DMKBIORS	Handles the resetting of a connection to the DASD Block I/O System Service, executable.
	DMSBIOSV	Handles pending SEVERs from a virtual machine to the DASD Block I/O System Service, executable.
DMKBLD		Pageable, executable.
	DMKBLDEC	Allocates storage for a virtual ECBLOK and the two TRQBLOKS required for a virtual machine with the ECMODE option, and initializes these blocks, executable.
	DMKBLDRL	Releases real segment, page, and swap tables to free storage, executable.
	DMKBLDRT	Creates and initializes segment, page, and swap tables as a function of virtual storage size, which is part of the process of building a users virtual machine, executable.
	DMKBLDVM	Creates and partially initializes a VMBLOK for a virtual machine, identified by its terminal real device block, executable.
DMKBOX		Pageable, non-executable.
		Provides the VM logo data, non-executable.
	DMKBOXMS	Provides the system minimum screen VM logo data, non-executable.
	DMKBOXNS	Provides the system normal screen VM logo data, non-executable.
	DMKBOXPR	Provides the system printer separator page logo data, non-executable.

Module Name	Entry Points	Attributes, Function
DMKBSC		Pageable, executable. Line error processing for remote 3270s on binary synchronous lines only.
	DMKBSCER	Examines the error condition resulting from a unit check or channel error that occurred while executing a CP-generated bisync line channel. DMKMSW is called to to notify the operator. After return from DMKMSW, the original channel program is terminated and the fatal flag is set in the IOBLOK. If the error is correctable, the channel program is re-executed up to a maximum of seven times, executable.
DMKCCD		Resident, executable.
	DMKCCDAS	Entry for non-dedicated DASD, executable.
	DMKCCDSK	Entry to insert SEEK command, executable.
DMKCCF		Resident, executable.
	DMKCCFBA	Entry for non-dedicated FBA, executable.
	DMKCCFBD	Entry for dedicated FBA, executable.
DMKCCH		Resident, executable. Operates with the I/O interrupt handler to schedule a device-dependent error recovery procedure when a channel data check, control check, or interface control check is detected.
	DMKCCHCF	Channel type table, non-executable.
	DMKCCHER	Continues channel check processing after obtaining the global system lock, executable.
	DMKCCHIS	Entry from DMKIOS when a channel check occurs when storing a CSW after a SIO, executable.
	DMKCCHMX	Save maximum size of CCH record, non-executable.
	DMKCCHNT	Entry from DMKIOTIN when a channel check occurs on an I/O interrupt, executable.
	DMKCCHRF	Entry from DMKDSP, DMKVIO, and DMKVSI to reflect channel check information to the virtual machine, executable.
	DMKCCHRT	Entry from DMKIOE to allow error messages to be printed, executable.
	DMKCCHSZ	Save the sizes of the IOEL, non-executable.
	DMKCCH60	Address of the 2860 channel module, non-executable.
DMKCCO		Resident, executable.
	DMKCCODD	Entry for dedicated DASD devices, executable.
	DMKCCOMS	Entry point for 3851 MSC devices, executable.
	DMKCCOTH	Other device class entry point, executable.
	DMKCCOTP	Entry for tape devices, executable.
DMKCCS		Resident, executable.
	DMKCCSEN	Handle sense commands, executable.
	DMKCCSLK	Obtain system lock, executable.
	DMKCCSRM	Get space routine, executable.
DMKCCT		Resident, executable.
	DMKCCTCN	Console entry point, executable.
	DMKCCTDL	Dialed device entry point, executable.
	DMKCCTLC	SDLC entry point, executable.
	DMKCCTRM	Terminal entry point, executable.

Module Name	Entry Points	Attributes, Function
DMKCCW		Resident, executable.
	DMKCCWCN	Get control information from user core, executable.
	DMKCCWCW	Write control data for other CCW translation modules, executable.
	DMKCCWGN	CP assist CCWGENRL instruction (E60F), executable.
		CP assist TRANLOCK instruction (E609), executable.
	DMKCCWRT	Return to common code in DMKCCW, executable.
	DMKCCWSB	Invokes an internal subroutine (CNTRLSUB) to obtain control bytes (seek data), executable.
	DMKCCWTR	Takes the list of virtual CCWs associated with the users SIO and translates it into a real CCW list, executable.
	DMKCCW0	CP assist DECCW0 instruction (E604), executable.
	DMKCCW1	CP assist DECCW1 instruction (E60C), executable.
DMKCDB		Pageable, executable.
		Processes DISPLAY, DCP commands.
	DMKCDBDC	Executes the DISPLAY command to display real storage locations, executable.
	DMKCDBDI	Displays virtual storage locations, storage keys, general registers, floating-point registers, PSW, CAW, and CSW at the terminal, executable.
DMKCDM		Pageable, executable.
		Processes DUMP and DMCP commands.
	DMKCDMDM	Dumps real storage to spooled printer.
		Dumps the contents of the specified real storage locations on the virtual printer spool file, executable.
	DMKCDMDU	Dumps the contents of the specified virtual storage locations, registers, PSW, and storage keys on the virtual printer spool file.
		Dumps virtual storage to the spooled printer, executable.
DMKCDS		Pageable, executable.
		Processes STORE and STCP commands.
	DMKCDSCP	Stores data into real storage (STCP command), executable.
	DMKCDSTO	Stores data into virtual storage (STORE command), executable.
DMKCFC		Pageable, executable.
		Gets the address of the routine that processes the CP console function that was requested.
	DMKCFCCO	Called by DMKUDRBV during system initialization to override the IBM-defined command and diagnose classes, executable.
	DMKCFCMD	Processes a CP console function, executable.
	DMKCFCTB	CP Command table, non-executable.
DMKCFD		Pageable, executable.
		Processes LOCATE and ADSTOP commands.
	DMKCFDAD	Stops virtual machine execution at specified address (ADSTOP command), executable.
	DMKCFDLO	Displays address of real device blocks, or VMBLOK and/or virtual device blocks (LOCATE command), executable.
DMKCFF		Pageable, executable.
	DMKCFFSB	Processes subroutines of DMKCFG, executable.

Module Name	Entry Points	Attributes, Function
DMKCFG		Pageable, executable.
	DMKCFGCL	Handles DIAGNOSE Code X'64', executable.
	DMKCFGII	Entry to IPL from LOGON (DMKLOG), executable.
	DMKCFGIP	Entry to IPL from a command line (DMKCFM), executable.
	DMKCFGIR	Handles re-IPL for a protected application user, executable.
DMKCFH		Pageable, executable.
	DMKCFHSV	Saves a virtual machines storage space including registers and PSW (SAVED System), executable.
	DMKCFHAS	If VMSAVE set to on it performs the same function as DMKCFHSV, executable.
DMKCFJ		Pageable, executable. Console function control.
	DMKCFJBE	Processes the BEGIN command. To allow the virtual machine to continue or to resume operation at a specified location, executable.
	DMKCFJRQ	Presents an attention interruption to the virtual machine to simulate a real request key interruption, executable.
	DMKCFJSL	Processes the SLEEP command. To allow terminal to receive messages, but not run the user's virtual machine, executable.
DMKCFM		Resident, executable. Processes DIAGNOSE Code 8. It scans the command line and goes to the required module.
	DMKCFMAT	Posts an attention interrupt pending for the virtual machine, executable.
	DMKCFMBK	Puts the terminal in console function (CP) mode (ATTN key pressed twice). Scans the command line and goes to the command handling routine, executable.
	DMKCFMEN	Entered when DIAGNOSE code 8 is executed or from DMKGRF to process immediate commands. Scans the command line and goes to the command handling routine, executable.
	DMKCFMRU	Entered to run the virtual machine, executable.
	DMKCFMSD	Executes a CP command that is in the text of the SEND command, executable.
	DMKCFMWU	Entered when sleep time expires, executable.
DMKCFO		Pageable, executable. Handles the SET command for authorized users.
	DMKCFOSA	Processes the SET SASSIST command, executable.
	DMKCFOSC	Processes the SET CPASSIST command, executable.
	DMKCFOSF	Processes the SET FAVORED command, executable.
	DMKCFOSP	Processes the SET PRIORITY command, executable.
	DMKCFOSQ	Processes the SET QDROP command, executable.
	DMKCFOSR	Processes the SET RESERVE command, executable.
	DMKCFOS3	Processes the SET S370E command, executable.
DMKCFP		Pageable, executable. Simulates the operators console for the virtual machine.
	DMKCFPRR	Handles system resets for other CP routines. Resets the virtual machine, executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKCFQ		Pageable, executable.
	DMKCFQRD	Handles virtual device reset for other CP routines, executable.
DMKCFR		Pageable, executable.
	DMKCFREP	Handles virtual device reset for other CP routines, executable.
DMKCFS		Pageable, executable.
		Processes the CP SET command for general users.
	DMKCFSAC	Processes the SET ACNT command, executable.
	DMKCFSAP	Processes the SET AUTOPOLL command, executable.
	DMKCFSAS	Enable/Disable VMSAVE entry point, executable.
	DMKCFSCC	Processes the SET CPCONIO command, executable.
	DMKCFSCP	Processes the SET CPUID command, executable.
	DMKCFSEC	Processes the SET ECMODE command, executable.
	DMKCFSEM	Processes the SET EMSG command, executable.
	DMKCFSIM	Processes the SET IMSG command, executable.
	DMKCFSIS	Processes the SET ISAM command, executable.
	DMKCFSLE	Processes the SET LINEDIT command, executable.
	DMKCFSMG	Processes the SET MSG command, executable.
	DMKCFSNT	Processes the SET NOTRANS command, executable.
	DMKCFSPX	Processes the SET PAGEX command, executable.
	DMKCFSRN	Processes the SET RUN command, executable.
	DMKCFSSA	Processes the SET SVCACCL command, executable.
	DMKCFSSM	Processes the SET SMSG command, executable.
	DMKCFSVC	Processes the SET VMCONIO command, executable.
	DMKCFSVS	Processes the SET VMSAVE command, executable.
	DMKCFSWG	Processes the SET WNG command, executable.
	DMKCFS37	Processes the SET 370E command, executable.
DMKCFT		Pageable, executable.
		Processes users' terminal options.
	DMKCFTRM	Entry point for the TERMINAL command processor. executable.
DMKCFU		Pageable, executable.
		Handles the SET commands for authorized users.
	DMKCFUDU	Processes the SET DUMP command, executable.
	DMKCFULO	Processes the SET LOGMSG command, executable.
	DMKCFUMI	Processes the SET MITIME command, executable.
	DMKCFUMO	Processes the SET MODE command, executable.
	DMKCFUPA	Processes the SET PAGING command, executable.
	DMKCFURE	Processes the SET RECORD command, executable.
DMKCFV		Pageable, executable.
	DMKCFVSB	Processes the SET STBYPASS command, executable.
	DMKCFVSM	Processes the SET STMULTI command, executable.
	DMKCFVMI	Processes the SET MIH command, executable.
DMKCFW		Pageable, executable.
	DMKCFWEP	To allow the user to modify the extended color and highlight attributes for his terminal, executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKCFY		Pageable, executable.
	DMKCFYAG	Processes the SET AFFINITY command for general users, executable.
	DMKCFYAS	Processes the SET AFFINITY command for primary system operators, executable.
	DMKCFYSA	Processes the SET ASSIST command, executable.
	DMKCFYSC	Processes the SET CONCEAL command, executable.
	DMKCFYSM	Processes the SET TIMER command, executable.
	DMKCFYSP	Processes the SET PF command, executable.
	DMKCFYPF	Allocates or releases the retrieve buffer, executable. Releases the PFKTABLE if not needed. SET AFFINITY, executable.
DMKCKD		Non-re-entrant, non-pageable, executable during initialization and shutdown. Checkpoint program device and processor handling routine.
	DMKCKDCP	External name, non-executable.
	DMKCKDEV	Checkpoint program device halting routine, executable.
	DMKCKDGP	Finds a usable path to a device, executable.
	DMKCKDSW	Handles requests to switch from one processor to another, executable.
DMKCKF		Non-re-entrant, non-pageable, executable during initialization and shutdown. System shutdown checkpoint routine.
	DMKCKFCN	Pointer to console list, non-executable.
	DMKCKFCV	Warm start cylinder I.D, non-executable.
	DMKCKFDL	Delete spool file chain anchor, non-executable.
	DMKCKFDV	Pointer to Dump device, non-executable.
	DMKCKFIL	Checkpoints the system onto the warm start cylinders during system shutdown, executable. (Saves open CPTRAP file when the system crashes.)
	DMKCKFLI	Beginning of list of addresses from DMKRSP, non-executable.
	DMKCKFLS	Size of list of addresses from DMKRSP, non-executable.
	DMKCKFOC	Pointer to system owned count, non-executable.
	DMKCKFOW	Pointer to system owned list, non-executable.
	DMKCKFRC	Pointer RECBLOK chain, non-executable.
	DMKCKFSF	SFBLOK size, non-executable.
	DMKCKFTE	Current end of the terminal buffer, non-executable.
	DMKCKFTP	Pointer to SYSRES type, non-executable.
	DMKCKFTR	Open CPTRAP spool file pointer, non-executable.
	DMKCKFVM	Pointer to system VMBLOK, non-executable.
	DMKCKFWA	Address of warm start INFO, non-executable.
	DMKCKFWM	Address of warm start cylinder, non-executable.
	DMKCKFWT	Address of DMKOPRWT, non-executable.

Module Name	Entry Points	Attributes, Function
DMKCKH		Non-re-entrant, non-pageable, executable during initialization and shutdown.
	DMKCKHAC	Saves printer or punch spool files which are active on the system printer or punch, or adds open spool file from the system reader to the delete queue, executable.
	DMKCKHDC	FBA read device characteristics buffer, non-executable.
	DMKCKHDL	Delimiter record, non-executable.
	DMKCKHIN	Index into the record buffer, non-executable.
	DMKCKHIO	Handles I/O interrupts, executable.
	DMKCKHPR	Handles read or write requests to and from the warm start area, executable.
	DMKCKHST	Checks validity of warm start cylinders, executable.
	DMKCKHSZ	Size of current records in record buffer, non-executable.
	DMKCKHWM	Marks warm start cylinders as valid, executable.
DMKCKM		Non-re-entrant, non-pageable, executable during initialization and shutdown.
	DMKCKMDV	Dump device address and RECBLOCK pointer, non-executable.
	DMKCKMSG	Displays messages for the checkpoint program, executable.
	DMKCKMSV	Saves multiple virtual machines during system shutdown or system abend, executable.
DMKCKN		Non-re-entrant, non-pageable, executable during initialization and shutdown. Checkpoint program service routines.
	DMKCKNBC	Beginning cylinder for EXT CCW define extent, non-executable.
	DMKCKNBH	Beginning head for EXT CCW define extent, non-executable.
	DMKCKNEC	Ending cylinder for EXT CCW define extent, non-executable.
	DMKCKNEH	Ending head for EXT CCW define extent, non-executable.
	DMKCKNIO	Routine to do I/O for the checkpoint program, executable.
	DMKCKNND	The end of the checkpoint program (not including the buffers), non-executable.
	DMKCKNPS	Buffer for swap table entries, non-executable.
	DMKCKNPW	Buffer to read in virtual machine pages, non-executable.
	DMKCKNPX	Buffer to read in DMKSNT, non-executable.
	DMKCKNPY	Storage key/register/PSW buffer, non-executable.
	DMKCKNPZ	Buffer of zeros for invalid pages, non-executable.
	DMKCKNRB	Warm start record buffer, non-executable.
	DMKCKNRD	Fixes up the RDEVBLOKS of all CP owned volumes for use by DMKCKM and DMKCKNIO, executable.
	DMKCKNTB	Enabled terminal buffer, non-executable.

Module Name	Entry Points	Attributes, Function
DMKCKP		Non-re-entrant, non-pageable, executable during initialization and shutdown. IPLed during the initial IPL, following a system ABEND dump, or to complete the execution of the shutdown command.
	DMKCKPCS	CSW for problem determination, non-executable.
	DMKCKPDV	Device address for problem determination, non-executable.
	DMKCKPER	Displays fatal error message and loads a disabled wait state PSW (code 007), executable.
	DMKCKPFL	Common flag byte used by checkpoint program, non-executable.
	DMKCKPIP	IPLed processor address, non-executable.
	DMKCKPLD	DASD address of DMKCKD, non-executable.
	DMKCKPLF	DASD address of DMKCKF, non-executable.
	DMKCKPLH	DASD address of DMKCKH, non-executable.
	DMKCKPLM	DASD address of DMKCKM, non-executable.
	DMKCKPLN	DASD address of DMKCKN, non-executable.
	DMKCKPMP	MP hardware present flag, non-executable.
	DMKCKPNC	Device class of SYSRES device, non-executable.
	DMKCKPND	DASD address of nucleus end, non-executable.
	DMKCKPRG	Loads a disabled wait state PSW (code 007), executable.
	DMKCKPRM	Running MP flag, non-executable.
	DMKCKPRS	DASD address of the first page of DMKSAV, non-executable.
	DMKCKPR2	DASD address of the second page of DMKSAV, non-executable.
	DMKCKPSN	Sense data for problem determination, non-executable.
	DMKCKPSR	SYSRES device address, non-executable.
	DMKCKPST	DASD address of DMKCKP, DASD address of the end of the checkpoint program, and the load point of DMKCKP, non-executable.
	DMKCKPT	IPL entry point for system initialization, system shutdown, and system abend, executable.
	DMKCKPTP	SYSRES device type, non-executable.
	DMKCKPVR	SAVEAREA for active V=R userid, non-executable.
DMKCKS		Pageable, executable. Performs checkpoint processing.
	DMKCKSIN	Initializes the check point cylinder after a successful warm start from the standard recovery procedure or after a cold start, executable.
	DMKCKSPL	Performs a checkpoint on any alterations in the spool file set up to allow the recovery routine to get them if warm start fails, executable.
DMKCKT		Pageable, executable.
	DMKCKTFS	Finds a slot on checkpoint cylinders for an SHQBLOK, SFBLOK, or an empty slot, executable.
	DMKCKTIN	Recovers spool file data if new RECBLOK needed, executable.
	DMKCKTML	Locates the specific slot map for SHQBLOKs and SFBLOKs, executable.
	DMKCKTMP	Gets a virtual page for and bring in the checkpoint slot maps, executable.
	DMKCKTSD	Deletes a spool fileid from the spool fileid bit map, executable.
	DMKCKTSF	Assigns an available spool fileid to an SFBLOK, executable.

Module Name	Entry Points	Attributes, Function
DMKCKV		Pageable, executable.
	DMKCKVWM	Recovers previously checkpointed spool file information. This information includes all open print or punch files in existence at the time the system went down or was shutdown. All open spool files are put in user hold status, executable.
DMKCLK		Pageable, executable. Synchronize the TOD clocks in an attached processor system.
	DMKCLKAP	Handles SYNC signal request, executable.
	DMKCLKCC	Handles CLKCHK signal request, executable.
	DMKCLKCK	Determines if the clock should be synchronized, executable. (Called from DMKCPI).
	DMKCLKMP	Synchronizes the clocks, executable.
	DMKCLKSC	Handles the TOD-sync-check external interrupt, executable.
DMKCMD		Pageable, non-executable. Contains tables which describe the syntax, entry points, and functional groups for subcommands. These tables enable DMKCFCMD to do centralized class validation for CP commands.
	DMKCMDAT	Subcommand table for attach (Type = R), non-executable.
	DMKCMDDE	Subcommand table for detach (Type = R), non-executable.
	DMKCMDDG	Subcommand table for detach (Type = G), non-executable.
	DMKCMDIN	Subcommand table for indicate (Type = A), non-executable.
	DMKCMDIG	Subcommand table for indicate (Type = G), non-executable.
	DMKCMDNO	Subcommand table for network (Type = O), non-executable.
	DMKCMDNR	Subcommand table for network (Type = R), non-executable.
	DMKCMDQA	Subcommand table for query (Type = A), non-executable.
	DMKCMDQG	Subcommand table for query (Type = G), non-executable.
	DMKCMDQP	Command entry for query PF processing, non-executable.
	DMKCMDQQ	Subcommand table for query (Type = P), non-executable.
	DMKCMDQR	Subcommand table for query (Type = R), non-executable.
	DMKCMDQS	Subcommand table for query (Type = S), non-executable.
	DMKCMDQV	Subcommand table for query virtual (Type = G), non-executable.
	DMKCMDSA	Subcommand table for set (Type = A), non-executable.
	DMKCMDSC	Subcommand table for set (Type = C), non-executable.
	DMKCMDSG	Subcommand table for set (Type = G), non-executable.
	DMKCMDSO	Subcommand table for set (Type = O), non-executable.
	DMKCMDSR	Subcommand table for set (Type = R), non-executable.
DMKCNS		Resident, executable. Real console terminal manager.
	DMKCNSED	Edits the input line for the following characters: escape, line end, line delete, and character delete, executable.
	DMKCNSEN	Enables or disables a low-speed terminal line, executable.
	DMKCNSIC	Entered from DMKQCN to initialize read and write CCWs for the CONTASK built by DMKQCN and DMKQCO, executable.
	DMKCNSIN	Interruption return point and handler for terminal I/O, executable.
	DMKCNSTB	Translate table for 'TRT' use, executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKCPB		Pageable, executable. Simulates the operators console for the virtual machine.
	DMKCPBEX	Processes the EXTERNAL command to present an external interruption to the virtual machine, executable.
	DMKCPBNR	Processes the NOTREADY command to cause the cause the virtual device to appear not ready, executable.
	DMKCPBRS	Processes the RESET command to reset all pending interrupts from the specified device, executable.
	DMKCPBRW	Processes the REWIND command to issue a rewind to the real tape device, executable.
	DMKCPBRY	Processes the READY command to simulate a device end interrupt to the specified device, executable.
	DMKCPBSR	Processes the SYSTEM command to simulate system reset and PSW restart to allow clearing of storage, executable.
DMKCPE		Resident, non-executable. Contains data constants that define the end of the CP nucleus.
	DMKCPEID	Base level identification of system, non-executable.
	DMKCPEML	CP assist level stored by DMKCPI, non-executable.
	DMKCPEND	At least one double word of free storage, non-executable.
	DMKCPEPD	Program product bit map - stored during initialization, non-executable.
DMKCPI		Pageable, executable.
	DMKCPIBD	Flag indicating bad frames were found, non-executable.
	DMKCPICA	CP assist flag byte, non-executable.
	DMKCPICD	Date and time when system was created, non-executable.
	DMKCPIFT	Address of the system file table, non-executable.
	DMKCPIF1	CPI AP flag, non-executable.
	DMKCPILF	Address of the last file table entry in use, non-executable.
	DMKCPIMS	Pointer to chain of delayed messages, non-executable.
	DMKCPINT	Initializes CP during IPL. Mainline system initialization routine (first half), executable.
	DMKCPINU	IPL volume file table entry, non-executable.
	DMKCPIOL	Warm start data flag, non-executable.
	DMKCPIRP	Temporary directory file pointer, non-executable.
	DMKCPISH	Shift, assist, and alternate path flag byte, non-executable.
	DMKCPISL	System name length, non-executable.
	DKMCPISN	System name, VM/SP, non-executable.
	DMKCPISW	Switch to issue message DMKOPE955W, non-executable.
	DMKCPITR	Flag indicating trace table is smaller than requested, non-executable.
	DMKCPIVP	Temporary override file pointer, non-executable.
	DMKCPIWC	Type of start flag byte, non-executable.
	DMKCPIWT	Number of BCTS which execute in 50 ms, non-executable.
DMKCPJ		Pageable, executable.
	DMKCPJNT	Mainline system initialization routine (second half), executable.
	DMKCPJST	Storage size, non-executable.
	DMKCPJTA	Trace table storage size, non-executable.

Module Name	Entry Points	Attributes, Function
DMKCPM		Pageable, executable.
	DMKCPMIO	Initializes all I/O paths for the processor being varied online, executable.
DMKCPN		Pageable, executable.
	DMKCPNAS	Assigns a 3480 tape device to the system, executable.
	DMKCPNVU	Processes the VARY ONLINE request, executable.
DMKCPO		Pageable, executable.
	DMKCPOFF	Processes the VARY OFFLINE PROCESSOR command, executable.
DMKCPP		Pageable, executable.
	DMKCPPUP	Releases all resources necessary for an AP/MP environment and reverts to uniprocessor mode, executable.
DMKCPS		Pageable, executable.
	DMKCPSH	Processes the HALT command, executable.
	DMKCPSSH	Processes the SHUTDOWN command, executable.
DMKCPT		Pageable, executable.
	DMKCPTNF	Processes the VARY command, executable.
DMKCPU		Pageable, executable.
	DMKCPUVY	Processes the VARY ONLINE PROCESSOR command, executable.
DMKCPV		Pageable, executable.
	DMKCPVAA	Punches user accounting records, executable.
	DMKCPVAC	Processes the ACNT command to create accounting records for logged-on users. Also, resets accumulated accounting information, executable.
	DMKCPVAE	Enables system low-speed lines for system restart, executable.
	DMKCPVDS	Processes the DISABLE command to disable an active line after the current user is finished with it, executable.
	DMKCPVEN	Processes the ENABLE command to enable the systems low-speed lines for system logon, executable.
DMKCPW		Pageable, executable.
	DMKCPWFB	Initiates IOBLOKs and builds RDCBLOKs, executable.
	DMKCPWVF	Varies a device offline, executable.
	DMKCPWUN	Unassigns a 3480 tape device from the system, executable.
DMKCPX		Pageable, executable.
	DMKCPXCK	Checks the number of t-disks to be released or cleared, executable.
	DMKCPXZT	Call DMKZTD to clear the t-disk, executable.
DMKCPY		Pageable, executable.
		Handles the lock and unlock commands.
	DMKCPYLK	Lock user pages into main storage, executable.
	DMKCPYUL	Unlock user pages previously locked, executable.
DMKCQC		Pageable, executable.
	DMKCQCPT	Processes the QUERY CPTRAP command, executable.

Module Name	Entry Points	Attributes, Function
DMKCQG		Pageable, executable. Processes the class G and class D Query commands.
	DMKCQGID	Processes the QUERY <VADDR>/<USERID> command, executable.
	DMKCQGQA	Processes the QUERY ALL command, executable.
	DMKCQGQC	Processes the QUERY CONSOLE command, executable.
	DMKCQGQD	Processes the QUERY DASD command, executable.
	DMKCQGQG	Processes the QUERY GRAF command, executable.
	DMKCQGQH	Processes the QUERY CHANNELS command, executable.
	DMKCQGQL	Processes the QUERY LINES command, executable.
	DMKCQGQS	Processes the QUERY STORAGE command, executable.
	DMKCQGQT	Processes the QUERY TAPES command, executable.
	DMKCQGQU	Processes the QUERY UR command, executable.
DMKCQH		Pageable, executable.
	DMKCQHFG	Processes QUERY FILES command for general users, executable.
	DMKCQHFI	Processes QUERY FILES command, executable.
	DMKCQHFS	Processes QUERY FILES command for spooling operators, executable.
	DMKCQHNG	Processes QUERY PUNCH command for general users, executable.
	DMKCQHNS	Processes QUERY PUNCH command for spooling operators, executable.
	DMKCQHRG	Processes QUERY READER command for general users, executable.
	DMKCQHRS	Processes QUERY READER command for spooling operators, executable.
	DMKCQHTG	Processes QUERY PRINTER command for general users, executable.
	DMKCQHTS	Processes QUERY PRINTER command for spooling operators, executable.
DMKCQP		Pageable, executable.
	DMKCQPQA	Processes the QUERY ALL command for system resource operators, executable.
	DMKCQPQD	Processes the QUERY DASD command for system resource operators, executable.
	DMKCQPQG	Processes the QUERY GRAF command for system resource operators, executable.
	DMKCQPQL	Processes the QUERY LINES command for system resource operators, executable.
	DMKCQPQP	Processes the QUERY PROCESSR command for system resource operators, executable.
	DMKCQPQS	Processes the QUERY STORAGE command for system resource operators, executable.
	DMKCQPQT	Processes the QUERY TAPES command for system resource operators, executable.
	DMKCQPQU	Processes the QUERY UR command for system resource operators, executable.
	DMKCQPSC	Processes the QUERY RADDR1-RADDR2 command, executable.

Module Name	Entry Points	Attributes, Function
DMKCQQ		Pageable, executable.
	DMKCQQID	Processes the QUERY userid and QUERY raddr commands for system resource operators, executable.
	DMKCQQQL	Processes the QUERY LINKS command for system resource operators, executable.
	DMKCQQQS	Processes the QUERY SYSTEM command for system resource operators, executable.
	DMKCQQQT	Processes the QUERY TDSK command for system resource operators, executable.
	DMKCQQPT	Displays real device path information as part of query command processing, executable.
DMKCQR		Pageable, executable.
	DMKCQRAF	Processes the QUERY AFFINITY command, executable.
	DMKCQRDP	Processes the QUERY DUMP command, executable.
	DMKCQRHD	Processes the QUERY HOLD command, executable.
	DMKCQRPG	Processes the QUERY PAGING command, executable.
	DMKCQRPR	Processes the QUERY PRIORITY command, executable.
	DMKCQRQD	Processes the QUERY QDROP command, executable.
DMKCQS		Pageable, executable.
	DMKCQSMI	Processes QUERY MITIME command, executable.
	DMKCQSNA	Processes QUERY NAMES command, executable.
	DMKCQSQC	Processes QUERY COMMAND command, executable.
	DMKCQSSC	Processes QUERY SCREEN command, executable.
DMKCQT		Pageable, executable.
	DMKCQTCL	Processes the QUERY CPLANG command, executable.
	DMKCQTRE	Constructs the messages in response to the QUERY GRAF command for all remote GRAF devices, executable.
	DMKCQTSN	Constructs the messages in response to the QUERY GRAF command for all SNA GRAF devices, executable.
	DMKCQTST	Processes the QUERY STATUS command for system resource operators, executable.
DMKCQU		Pageable, executable. Handles QUERY command for system resources.
	DMKCQUSE	Processes the QUERY SET command, executable.
	DMKCQUST	Processes the QUERY TERMINAL command, executable.
	DMKCQUTE	Processes the QUERY SECUSER command, executable.
DMKCQY		Pageable, executable.
	DMKCQYCA	Processes the CPASSIST command, executable.
	DMKCQYCL	Processes the CPLEVEL command, executable.
	DMKCQYCP	Processes the CPUID command, executable.
	DMKCQYID	IPL date SAVEAREA (DMKCPI), executable.
	DMKCQYIT	IPL time SAVEAREA (DMKCPI), executable.
	DMKCQYLM	Processes the LOGMSG command, executable.
	DMKCQYPF	Processes the PF command, executable.
	DMKCQYRG	Displays page range of a saved VMSAVE system, executable.
	DMKCQYSA	Processes the SASSIST command, executable.
	DMKCQYSP	Processes the SPMODE command, executable.
	DMKCQYS3	Processes the S370E command, executable.
	DMKCQYTI	Processes the TIME command, executable.
	DMKCQYUI	Processes the USERID command, executable.
	DMKCQYUS	Processes the USERS command, executable.
	DMKCQYVS	Processes the VMSAVE command, executable.

Module Name	Entry Points	Attributes, Function
DMKCRM		Pageable, executable.
	DMKCRMCN	Handles pending connections to the Collection Resource Management System Service, executable.
	DMKCRMIL	Handles pending messages received by the Collection Resource Management System Service from the TSAF virtual machine, executable.
	DMKCRMQS	Handles pending QUIESCEs from the TSAF virtual machine on the *CRM path, executable.
	DMKCRMSV	Handles pending severs by the TSAF virtual machine on the *CRM path, executable.
DMKCSB		Pageable, executable.
	DMKCSBLD	Processes the LOADBUF command (real UCS or FCB buffer), executable.
	DMKCSBLF	Reloads the last FCB buffer, executable.
	DMKCSBSP	Space to first line of next form, executable.
	DMKCSBVL	Processes the LOADVFCB (load virtual forms control buffer) command, executable.
DMKCSC		Pageable, executable.
	DMKCSCLD	Obtains the appropriate CCWs and points to the correct data module for loading the real printers UCS/FCB buffer. Searches for the requested UCS/FCB image in the data module. This entry point will be called for real printer buffer loading, executable.
	DMKCSCLV	Searches through module DMKFCB for the requested FCB image, which will be loaded into the virtual FCB buffer. This entry point will be called for virtual printer buffer loading, executable.
DMKCSF		Pageable, executable.
		Processes real spooling commands for real unit record devices, executable.
	DMKCSFBS	Processes the BACKSPACE command, executable.
	DMKCSFFL	Processes the FLUSH command, executable.
	DMKCSFRP	Processes the REPEAT command, executable.
	DMKCSFSP	Processes the SPACE command, executable.
DMKCSO		Pageable, executable.
		Processes real spooling commands for real unit record devices.
	DMKCSODR	Processes the DRAIN command, executable.
	DMKCSOSD	Restarts a device after it has been drained, executable.
	DMKCSOST	Processes the START command by device type, executable.
DMKCSP		Pageable, executable.
	DMKCSPSP	Processes the SPOOL command, executable.
DMKCSQ		Pageable, executable.
	DMKCSQCL	Processes the CLOSE command, executable.
	DMKCSQFR	Processes the FREE command, executable.
	DMKCSQHL	Processes the HOLD command, executable.
DMKCSR		Pageable, executable.
	DMKCSRGT	Searches the device type from the SPOOL command, executable.
	DMKCSRPO	Processes the SPOOL command from general users and spooling operators, executable.

Module Name	Entry Points	Attributes, Function
DMKCST		Pageable, executable. Processes class G commands.
	DMKCSTAG	Entry point to process the TAG command, executable.
DMKCSU		Pageable, executable.
	DMKCSUCG	Processes the CHANGE command for general users, executable.
	DMKCSUCS	Processes the CHANGE command for spooling operators, executable.
DMKCSV		Pageable, executable.
	DMKCSVOG	Processes the ORDER command for general users, executable.
	DMKCSVOS	Processes the ORDER command for spooling operators, executable.
	DMKCSVPG	Processes the PURGE command for general users, executable.
	DMKCSVPS	Processes the PURGE command for spooling operators, executable.
DMKCSW		Pageable, executable.
	DMKCSWCH	Processes the CHANGE command for general users and spooling operators, executable.
DMKCSX		Pageable, executable.
	DMKCSXTG	Processes the TRANSFER command for general users. executable.
	DMKCSXTS	Processes the TRANSFER command for spooling operators. executable.
DMKCVT		Resident, executable. Processes the conversion routines.
	DMKCVTAB	External entry to force CVT001 abend, executable.
	DMKCVTBD	Converts a word of binary data into a doubleword of decimal digits, executable.
	DMKCVTBH	Converts a word of binary data into a doubleword of hexadecimal data, executable.
	DMKCVTDB	Converts a decimal field into a fullword of binary data, executable.
	DMKCVTDT	Converts data and time to EBCDIC and inserts it into a specified location, executable.
	DMKCVTHB	Converts the designated hexadecimal field into a binary fullword, executable.
DMKCVU		Pageable, executable.
	DMKCVUFP	Converts a floating-point doubleword into 17 bytes of decimal data, executable.
DMKDAD		Resident, executable.
	DMKDADER	Processes 3375/3380 error recovery for CP and DIAGNOSE I/O, executable.
DMKDAS		Resident, executable. DASD error retry program.
	DMKDASER	Retries the failing DASD channel program, executable.
DMKDAU		Resident, executable. FB-512 error retry program.
	DMKDAUER	Retries the failing FB-512 channel program, executable.

Module Name	Entry Points	Attributes, Function
DMKDDR		Residency not applicable, executable. This is the DASD dump restore program. It saves data from a direct access volume onto a tape or tapes. It returns data to DASD from tape that has been placed on the tape by this program. It copies data from one device to another of the same type. It prints a translation of each record specified on the SYSPRINT device. Prints a translation of each record specified on the console. Initial program loaded or run under CMS if on a CMS disk.
	DMKDDREP	DASD dump restore program entry point, executable.
	DMKDDRED	End-of-load module for CMS, non-executable.
DMKDEF		Pageable, executable. Processes the DEFINE command to define a virtual device.
	DMKDEFDG	Processes the DEFINE command for general users, executable.
	DMKDEFDS	Processes the DEFINE command for system resource operators, executable.
DMKDEG		Pageable, executable.
	DMKDEGIN	Processes the DEFINE STORAGE and DEFINE CHANNEL commands, executable.
DMKDEI		Pageable, executable.
	DMKDEIMS	Defines or redefines MSS disks, executable.
DMKDGD		Resident, executable. Processes simple disk I/O.
	DMKDGDDK	Performs simple disk I/O of a standardized format with a minimum of CCW chain manipulation, executable.
	DMKDGDUL	Unlocks user pages upon completion of I/O, executable.
DMKDGF		Resident, executable.
	DMKDGFIN	Handles simple disk interruptions, executable.
DMKDIA		Pageable, executable.
	DMKDIAIR	Processes return from resetting device I/O, executable.
	DMKDIAL	Processes the DIAL command. Attaches a users terminal as a dedicated device to an existing virtual 270X terminal line in the virtual machine addressed by the command line, executable.

Module Name	Entry Points	Attributes, Function
DMKDIB		Pageable, executable.
	DMKDIBCP	COUPLE command processor. Establishes a virtual connection between two channel-to-channel adapters on a single virtual machine, executable.
	DMKDIBDR	Releases a terminal line that has been in use by the virtual machine via the DIAL command. The line is detached from the virtual machine and made available for normal log on to the system, executable.
	DMKDIBSM	Simulates sense data and status for virtual I/O to a simulate device (2702 line or CTCA) that has not yet been activated through either the console function DIAL for 2702 lines, or the console function COUPLE for virtual CTCAs, executable.
DMKDID		Resident, executable. Handles missing interruptions.
	DMKDIDDA	Examines RDEVBLOKs for DASD, executable.
	DMKDIDEP	Scans all RDEVBLOKs to identify missing interrupts. IF flags RDEVBUZY and RDEVMID are one, an operator message is issued, non-executable.
	DMKDIDGR	Examines RDEVBLOKs for graphic devices, except 1053 and 328X printers, executable.
	DMKDIDLF	Called via SVC during LOGOFF/FORCE processing when necessary to clean up outstanding I/O requests to attached tape drives or the virtual machine console, executable.
	DMKDIDMS	Examines RDEVBLOKs for miscellaneous devices, executable.
	DMKDIDTA	Examines RDEVBLOKs for tape devices, executable.
	DMKDIDTR	Contains a list of TRQBLOK addresses, non-executable.
	DMKDIDUR	Examines RDEVBLOKs for unit record devices, except 3800 and 3289E printers, executable.
DMKDIF		Pageable, executable. Completes the DIAL processing started in DMKDIA.
	DMKDIFDI	Completes the dedication of a device to a virtual machine, executable.
DMKDIR		Standalone, executable. Initial program loaded or run under CMS if on a CMS disk.
	DMKDIRCT	Builds a user directory on a system owned volume using pre-allocated cylinders, executable.
	DMKDIRED	End of load module for CMS, non-executable.

Module Name	Entry Points	Attributes, Function
DMKDMP		Resident, executable. Writes a dump of main storage, control registers, floating-point registers, general registers, and clocks to a specified device.
	DMKDMPAL	Dump allocation flag byte, non-executable.
	DMKDMPAS	Dump taken to this device, non-executable.
	DMKDMPAU	Pointer to automatic dump device RDEVB, non-executable.
	DMKDMPBG	Non-executable.
	DMKDMPCA	Checkpoint device address, non-executable.
	DMKDMPDS	Addressability pointer for DSECT macro to insert common DCs, non-executable.
	DMKDMPDK	Writes the dump on the specified device, executable.
	DMKDMPDT	Date - MM/DD/YY - edited EBCDIC, non-executable.
	DMKDMPRC	Pointer to RECBLOK chain for automatic dump device, non-executable.
	DMKDMPRS	Initial program loads the system over again, executable.
	DMKDMPSD	System IPL device, non-executable.
	DMKDMPSF	SFBLOK for automatic dump device, non-executable.
	DMKDMPTD	TOD clock at 00.00.00 today, non-executable.
DMKDRD		Pageable, executable. Process spool files
	DMKDRDDD	Deletes system dump spool file, executable.
	DMKDRDER	Manipulates input spool files via a DIAGNOSE Code X'0014' issued by the virtual machine, executable.
	DMKDRDMP	Reads a system dump spool file via a DIAGNOSE Code X'0034' issued by the virtual machine, executable.
	DMKDRDSY	Reads the system symbol table CSECT via a DIAGNOSE Code X'0038' issued by the virtual machine, executable.
DMKDSB		Resident, executable. DASD error retry program.
	DMKDSBRD	Processes unsolicited device end interruptions, executable.
	DMKDSBSD	Collects DASD sense data, executable.

Module Name	Entry Points	Attributes, Function
DMKDSP		Resident, executable. Entered after each interruption handler is finished processing and after each stacked CPEXBLOK, I/O request, and external interruption has been serviced. It updates the CPU times charged to the user that has received service, updates all virtual timers, and reflects any pending interruptions for which the user is enabled. After the user's status has been updated, the highest-priority runnable user is dispatched.
	DMKDSPA	Immediate redispatch path for virtual machines The only status update that occurs is for virtual timers, executable.
	DMKDSPAC	Count of DMKDSPA entries, non-executable.
	DMKDSPB	Processes new virtual PSW and dispatch. Entered if the virtual PSW has been entered outside DMKDSP, executable.
	DMKDSPBC	Count of DMKDSPB entries, non-executable.
	DMKDSPCC	Count of DMKDSPCH entries, non-executable.
	DMKDSPCH	Main entry point. Updates timers and dispatches user, executable.
	DMKDSPCK	Virtual clock comparator reflection counter, non-executable.
	DMKDSPE	Processes interrupt from virtual interval timer.
	DMKDSPEC	Count of DMKDSPE entries, non-executable.
	DMKDSPIT	Virtual interval timer reflection counter, non-executable.
	DMKDSPNP	Number of dynamically assignable page frames now available in the system, non-executable.
	DMKDSPPT	Virtual CPU timer reflection counter, non-executable.
	DMKDSPQI	Base for queue of QVM interrupts, non-executable.
	DMKDSPQS	Dispatched users maximum time slice, non-executable.
	DMKDSPRC	Count of DMKSPRU entries, non-executable.
	DMKDSPRQ	Queues anchor for IOBLOKs and CPEXBLOKs, non-executable.
	DMKDSPRU	Entered in attached processor mode when the system lock is not held, executable.
	DMKDSPSM	Standby virtual mode assist control byte, non-executable.
	DMKDSPVM	Flags for QVM and user VMBLOK address, non-executable.
	DMKDSPWI	Address of enable window for DMKPSA/DMKEXT, non-executable.
	DMKDSP0	CP assist DSP0 instruction (E60D), non-executable.
	DMKDSP1	CP assist DSP1 instruction (E607), non-executable.
	DMKDSP2	CP assist DSP2 instruction (E611), non-executable.
DMKEIG		Pageable, executable.
	DMKEIG80	Analyzes the 2880 channel logout and sets appropriate bits in the ECSW field according to the results of this analysis. It moves the channel logout to the channel check record, executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKENT		Resident, executable. Meets the CP nucleus residency requirements for TRQBLOK and CPEXBLOK entries to pageable VM. Monitors module DMKMIA.
	DMKENTBS	Status of start TRQ, non-executable.
	DMKENTEC	Used to invoke a MONITOR STOP command via a CPEXBLOK, executable.
	DMKENTES	TRQ for stopping monitoring, non-executable.
	DMKENTET	Used to invoke a MONITOR STOP command via a TRQBLOK request, executable.
	DMKENTFI	Used to complete monitor shutdown processing, via CPEXBLOK, executable.
	DMKENTKC	Used to invoke a MONITOR CLOSE command via a CPEXBLOK, executable.
	DMKENTSC	Used to invoke a MONITOR START SPOOL command via a CPEXBLOK, executable.
	DMKENTSK	Pointer to SEEKS list, non-executable.
	DMKENTST	Used to invoke a MONITOR START SPOOL command via a TRQBLOK request, executable.
	DMKENTTB	TRQ for starting monitor, non-executable.
	DMKENTTE	Status of end TRQ, non-executable.
	DMKENTTI	High frequency I/O status sampling routine, entered every two seconds via TRQBLOK request, executable.
	DMKENTUT	I/O utilization interval, non-executable.
	DMKENT62	DASTAP class 6 code 2 I/O status sampling routine, called from DMKMONTI every 60 seconds to collect the data accumulated by DMKENTTI, executable.
	DMKENT63	Collect I/O initialization data, non-executable.
DMKEPS		Pageable, executable.
	DMKEPSWD	Prompts the user to enter a password, types masking characters if appropriate, reads the password from the terminal, and checks it for a match, executable.
DMKERM		Pageable, executable.
	DMKERMCP	Edit and display system error messages and responses, executable.
	DMKERMSG	Accesses the requested message from the CP message repository and inserts the module ID, message number, and data. It then prints the message, executable.
DMKEXT		Resident, executable.
	DMKEXTIN	Entry point for External Interrupt handler, executable.
	DMKEXTSP	Handles all SIGNAL actions after CP initialization, executable.
	DMKEXTSL	Entry point for External Interrupt handler, executable.
	DMKEXTST	Logical address of stopped processor, non-executable.
DMKFCB		Pageable, non-executable.
	DMKFCBLD	Contains the forms control load buffer that the LOADBUF command uses to load the forms control buffer in the 3811 control unit for the 3203 or 3211 printer. The LOADVFCB command also uses DMKFCB to load the forms control buffer in the virtual 3203 or 3211 printer, non-executable.

Module Name	Entry Points	Attributes, Function
DMKFMT		Standalone program.
	DMKFMT	Accepts parameters from the console or IPL device (card reader) and performs partial or complete formatting, allocating, and labeling of 2314, 2319, 3330, 3340, 3350, 3375, 3380, and 2305 DASD, as well as any FB-512 devices. The FORMAT program also write-checks the surfaces. Bad surfaces are flagged to prevent their use. No alternative tracks are assigned. OS labels are written to be compatible with OS, but labels indicate to OS that no space is left on the DASD device. All input parameters are verified for correctness.
DMKFPS		Resident, executable.
	DMKFPS	Fast privileged simulation for selected privileged operations.
DMKFRE		Resident, executable.
		Free storage management module.
	DMKFREAP	Back pocket for AP mode extend, non-executable.
	DMKFREDE	Entry via CPEXBLOK to resume deferred free call when running AP/MP, executable.
	DMKFREE	Gets space from free storage and processes the CP assist FREE instruction (E600), executable.
	DMKFREHI	Beginning of high-core free storage area, non-executable.
	DMKFRELG	External name for LRGSTSIZ, non-executable.
	DMKFRELN	Number of blocks in DMKFRELS chain, non-executable.
	DMKFRELO	End of low-core free storage area, non-executable.
	DMKFRELS	Start of large block storage chain, non-executable.
	DMKFREMT	External name for common area for DMKFRT, non-executable.
	DMKFREMX	Maximum subpool size in doublewords, non-executable.
	DMKFRENP	External name for NPAGFREE and NPAGFRET, non-executable.
	DMKFRERC	Special entry point to acquire free storage. If the storage request cannot be satisfied, a condition code of one is returned to the caller, executable.
	DMKFRESC	Split counters for DMKMON, non-executable.
	DMKFREST	Subtable - Pointers to subpools, non-executable.
	DMKFRESV	SAVEAREA for use when calling DMKPTRFR, non-executable.
	DMKFRESW	Entry via CPEXBLOK for extend processing after execution is transferred back to the processor initiating the extend (AP/MP only), executable.
	DMKFRES2	Second SAVEAREA for PTRFR call (AP/MP), non-executable.
	DMKFRET	Processes the CP Assist FRET instruction (E601) to return space to free storage, executable.
	DMKFRETL	External name for the 45 byte data list used by the CP assist, non-executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKFRT		Resident, executable.
		Free storage management module.
	DMKFRTRS	Returns subpools to free storage chain, executable.
	DMKFRTSN	Scans the free storage chain to return page frame sized blocks back to the dynamic paging area, executable.
	DMKFRTT	Returns storage to a subpool or the large storage chain. Handles requests to return storage that cannot be handled by the CP Assist Fret instruction at DMKFRET, executable.
	DMKFRTTE	Returns storage to the large storage chain. This entry is called by DMKUSP to return storage blocks known to have been obtained from the dynamic paging area, executable.
	DMKFRTTR	Assigns storage to free storage management; does not release pages, executable.
DMKGIO		Pageable, executable.
		Initializes supervisor operations for tape, unit record, and nonstandard disk I/O operations.
	DMKGIOEX	Checks device validity and initializes I/O operations on tape, unit record, and nonstandard disk I/O programs per supervisor call. This module presents resultant condition code and CSW (if warranted) to the user, executable.
DMKGRA		Pageable, executable.
	DMKGRAOT	Performs APL/TEXT translations for outbound 3270 data, executable.
DMKGRC		Resident, executable.
	DMKGRCCP	Space for system ID, non-executable.
	DMKGRCHL	Space for system ID, non-executable.
	DMKGRCMR	Space for system ID, non-executable.
	DMKGRCQR	Process the query response from remote BSC devices, executable.
	DMKGRCQY	Determines if terminal or its controller has extended data stream capability, executable.
	DMKGRCRU	Space for system ID, non-executable.
	DMKGRCUP	Generates the data stream necessary to update the 3270 screen, according to parameters input in register 2, executable.
	DMKGRCVM	Space for system ID, non-executable.
	DMKGRCWC	Processes a write CONTASK, executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKGRF		Resident, executable. Supports local 3270, 3278 Model 2A, and 3279 devices. Processes interruptions and CCWs for the devices, including message handling and screen management.
	DMKGRFEN	Enables or disables the device.
	DMKGRFIC	Starts a CONTASK from DMKQCN. If a larger than 4K buffer is required, a CCW is built indicating that indirect addressing will be used.
	DMKGRFIN	Handles the interruption via an IOBLOK, executable.
	DMKGRFMT	Address of contask/logo buffer and logo size as initialized by DMKGRTFM, executable.
.	DMKGRFTI	Processes clock comparator timer interrupts, executable.
DMKGRH		Resident, executable.
	DMKGRHIN	Processes channel programs for 3066 display, executable.
DMKGRT		Resident, executable. Contains common data areas and subroutines for 3270 display support.
	DMKGRTAB	Computes the next tab position and creates the data stream to position the cursor and insert a logical tab character if necessary, executable.
	DMKGRTAC	Counts entries in AID table, non-executable.
	DMKGRTAI	Accesses total AID table, non-executable.
	DMKGRTB	Orders for 327x Model 2 display terminal, executable.
	DMKGRTDT	Edit data from input fields in logo screen, executable.
	DMKGRTFD	VM/SP message, non-executable.
	DMKGRTFM	Brings in the VM/SP logo and initializes buffer and the CCWs to write the logo in DMKGRF and DMKRGB, executable.
	DMKGRTFO	Header for VM/SP message, non-executable.
	DMKGRTPF	Accesses PF Key portion of the AID table, non-executable.
	DMKGRTP6	Accesses table entry for PF6, non-executable. This is used for the PA 3 key.
	DMKGRTT1	Allow user to logon again after being delayed by logon inductor, executable.
	DMKGRT12	Screen coordinate conversion table, non-executable.
DMKHPS		Pageable, executable.
	DMKHPSDG	Handles the graphic communication DIAGNOSE, executable.
	DMKHPSHT	Executes virtual HALT I/O, executable.
	DMKHPSQR	Handles an I/O operation directed to a logical device by CP, executable.
	DMKHPSQV	Handles an I/O operation directed to a logical device by a virtual machine, executable.
DMKHPT		Pageable, executable.
	DMKHPTDI	Terminates a logical device, executable.
	DMKHPTEX	Reflects special external interrupt, executable.
	DMKHPTRE	Resets all logical devices owned by a virtual machine, executable.
DMKHPU		Pageable, executable.
	DMKHPUSR	3270 Logical Device Support subroutines, executable.

Module Name	Entry Points	Attributes, Function
DMKHVC		Resident, executable.
	DMKHVCAL	Contains a table which it searches to scan and validate all DIAGNOSE instructions. It processes the DIAGNOSE instruction locally or passes control as needed to DMKHVD or DMKHVE, executable.
	DMKHVCDE	End of all user class definition tables, non-executable.
	DMKHVCDG	System diagnose class table, non-executable.
	DMKHVCDI	Count of diagnose I/Os, non-executable.
	DMKHVCDU	Installation defined diagnose classes, non-executable.
	DMKHVCPC	DASD pages/cylinder table, non-executable.
	DMKHVCVI	Diagnose X'28' entry for DMKVIO, executable.
DMKHVD		Pageable, executable.
	DMKHVDAL	Performs services for virtual machines as requested by the DIAGNOSE instruction, executable.
	DMKHVDIP	Internal entry for DIAGNOSE X'0040', executable.
	DMKHVDPP	Initialize program product bit map, executable.
DMKHVE		Pageable, executable.
	DMKHVEAL	Performs DIAGNOSE X'2C', X'30', X'8C', X'94' and X'98' for the virtual machine, executable.
	DMKHVEPC	Data table, DASD pages/cylinder, executable.
	DMKHVEYL	Data table, DASD cylinders/device, executable.
DMKHVF		Pageable, executable.
	DMKHVFAL	Performs DIAGNOSE codes X'C8', X'CC', and X'D4' for the virtual machine, executable.
	DMKHVFCR	Locates or creates a specified LANGBLOK, executable.
DMKIDR		Pageable, executable.
	DMKIDRCN	Handles pending connections from a virtual machine to the Identify System Service, executable.
	DMKIDRFN	Handles requests to find the resource owner received from APPC/VM processing of connections to a resource name, executable.
	DMKIDRFX	Cleans up pending messages sent to the TSAF virtual machine when the *CRM path has been severed, executable.
	DMKIDRIN	Handles revoke messages from the TSAF virtual machine passed to the Identify System Service by the Collection Resource Management System Service, executable.
	DMKIDRSV	Handles severs of the path to the Identify System Service, executable.
DMKIDU		Pageable, executable.
	DMKIDUMP	Initializes DASD dump cylinders, executable.
	DMKIDUSF	Assigns a spool file ID to the dump SFBLOK, executable.
	DMKIDUTD	Anchor for TDKBLOK chain, non-executable.
	DMKIDUTU	Count of temporary pages in system, non-executable.
DMKIMG		Pageable, executable.
	DMKIMGBG	Provides a CMS interface for a VS-based IEBIMAGE program, executable.
DMKIOC		Pageable, executable.
	DMKIOCVT	Converts VM/SP device type to OS/VS device type, executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKIOE		Resident, executable. This is the error recording module. It receives all requests for error recording and passes control to the proper pageable routine after checking if a recording is in progress. If a previous request for error recording is in progress, the current request is queued on the appropriate queue for recording at a later time. It makes checks to see if the recording cylinder is full. DMKIOE also interfaces with the pageable module that initializes and erases the error recording cylinders.
	DMKIOECC	Entry for a channel error condition occurring on a SIO in DMKIOS with a response condition code of one, executable.
	DMKIOECE	CECYL, non-executable.
	DMKIOECL	SYSRES device class (used in DMKIOF and DMKIOG), non-executable.
	DMKIOECQ	Record queue chain for Channel Check records, non-executable.
	DMKIOECT	Number of error recording cylinders, non-executable.
	DMKIOEEP	Error recording CCP (CYL + Page), non-executable.
	DMKIOEES	Recording area full flag, non-executable.
	DMKIOEFL	Entry point to locate the starting page record for recording, executable.
	DMKIOEFM	Entry to clear and format the recording area on disk, executable.
	DMKIOEFR	Indication of whether frames exist on error recording cylinders, non-executable.
	DMKIOEHS	Address of first error record (CCPD) or place where it should be written if none is there now (LOGREC HDRSTART value), non-executable.
	DMKIOEIF	Infoflag, non-executable.
	DMKIOEIQ	Record queue chain for OBR/MDR records, non-executable.
	DMKIOEIR	Pointer to intensive recording block, non-executable.
	DMKIOEMC	Entry for machine check recording, executable.
	DMKIOEMI	Builds and formats the missing interrupt record, executable.
	DMKIOEMQ	Record queue chain for Machine Check records, executable.
	DMKIOEMX	Maximum 'CCP' or 'PPP' in area, non-executable.
	DMKIOENI	90% point in record area, non-executable.
	DMKIOENQ	Record queue chain for environmental records, executable.
	DMKIOERN	Processes a 37xx and remote 3270 request, executable.
	DMKIOERP	Recording in progress flag, non-executable.
	DMKIOERQ	Queue chain for erase requests, non-executable.
	DMKIOERR	Schedules recording for unit check, channel errors, executable.
	DMKIOESD	Format hardware environmental counters, executable.
	DMKIOESQ	Queue chain for update requests, non-executable.
	DMKIOESR	Handle OBR record requests synchronously, executable.
	DMKIOEST	Schedules the update of a statistical data request, executable.
	DMKIOETY	SYSRES device type (used for paging), non-executable.
	DMKIOEVR	Processes an SVC 76 request and MIH request, executable.
	DMKIOEVQ	Record queue chain for SVC 76 records, non-executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKIOF		Pageable, executable. Records system and I/O errors on the system disk in predefined error recording cylinders.
	DMKIOFCN	Handles CONNECTs from a virtual machine to the Error Logging System Service, executable.
	DMKIOFC1	Records channel check error from SIO in DMKIOS when cc=1, executable.
	DMKIOFIN	Initializes pointers to available recording pages at IPL and after an erase has been completed, executable.
	DMKIOFIU	Send log records via IUCV to virtual machine, executable.
	DMKIOFM1	Record machine checks records, executable.
	DMKIOFOB	Record Outboard and Miscellaneous data records, executable.
	DMKIOFSV	Handles SEVERs from a virtual machine to the Error Logging System Service, executable.
	DMKIOFVR	Records errors when requested by SVC 76, executable.
DMKIOG		Pageable, executable. Called at initialization to locate the error recording device, locate the last outboard error record and system recordings made on the cylinders, and set the in-storage pointers to the correct values. Initialization for RMS functions is performed after first making a test to determine if CP is running under CP. RMS functions are not activated for a virtual CP environment. This module also erases the recording areas.
	DMKIOGAP	Initialization entry, RMS initialization for attached processor, executable.
		Contains all function of DMKIOG except erase, executable.
	DMKIOGF1 DMKIOGF2	Erases (1) error records or (2) error records and frame records from the error recording cylinders, depending on input parameters, executable.
DMKIOH		Pageable, executable. Called at initialization if the system error recording area requires formatting, or called while processing a CPEREP CLEARF request. Reads and formats machine check and channel check frames obtained from an SRF device (7443). Recognizes multiple SRF devices in an attached processor environment. Attempts to read frames from each available SRF device that was generated and format the respective frames at the beginning of the error recording cylinders.
	DMKIOHFR	For 3031/3032/3033 processors, reads frames from SRF devices, formats them in 4096-byte blocks, and writes the records to the error recording cylinders. The appropriate CPU id data is stored in the header portion of each frame record formatted by DMKIOHFR, executable.
DMKIOJ		Pageable, executable.
	DMKIOJBL	Builds and formats outboard and miscellaneous data records, executable.

Module Name	Entry Points	Attributes, Function
DMKIOQ		Resident, executable. IOBLOK queue handler and device path finder.
	DMKIOQDE	Dequeues the next IOBLOK from a real device queue, executable.
	DMKIOQDH	Dequeues the next IOBLOK from a real channel queue, executable.
	DMKIOQDQ	Subroutine to dequeue mini-IOBLOKs from RBLOKs and unchain mini-IOBLOKs from other IOBLOKs, executable.
	DMKIOQDU	Dequeues the next IOBLOK from a real control unit queue, executable.
	DMKIOQFC	Finds a channel path to the device, executable.
	DMKIOQFP	Finds a path to the device, executable.
	DMKIOQFX	Finds a fixed path to the device, executable.
	DMKIOQQD	Queues an IOBLOK on a real device queue, executable.
	DMKIOQQU	Subroutine to queue control unit busy IOBLOK, executable.
	DMKIOQSK	Queues an IOBLOK on a real channel queue, executable.
	DMKIOQUS	Queues an IOBLOK on a real control unit queue, executable.
DMKIOS		Resident, executable. Schedules requests for virtual machine and program I/O operations.
	DMKIOSCB	Control unit busy queue, non-executable.
	DMKIOSC1	Handles a deferred condition code 1 from SIOF, executable.
	DMKIOSC3	Handles a deferred condition code 3 from SIOF, executable.
	DMKIOSEN	Handles sense operations after a unit check, executable.
	DMKIOSER	Error recovery processing, executable.
	DMKIOSHA	Halts an active device, executable.
	DMKIOSMQ	Mini-IOBLOK queue, non-executable.
	DMKIOSNM	Count of depleted mini-IOBLOK stack, non-executable.
	DMKIOSQE	Reschedules an I/O operation after an I/O error on the channel, executable.
	DMKIOSQR	Schedules a CP-generated I/O operation, executable.
	DMKIOSQV	Schedules a virtual machine I/O operation, executable.
	DMKIOSRC	Error recording, executable.
	DMKIOSRH	Restarts channel, executable.
	DMKIOSRQ	Requeues request, executable.
	DMKIOSRS	Restarts device, executable.
	DMKIOSRU	Restarts control unit-channel, executable.
	DMKIOSRW	Processes the IOBLOK used for REWIND, executable.
	DMKIOSST	Issues SIO, executable.
	DMKIOSW1	Restart channel (referenced by DMKDSP), non-executable.
DMKIOT		Resident, executable.
	DMKIOTCT	Count of total I/O interrupts, non-executable.
	DMKIOTIN	Processes all I/O interruptions, executable.
	DMKIOTRC	Processes all pseudo I/O interrupts generated by DMKACRCV, executable.
DMKISM		Pageable, executable.
	DMKISMTR	Finds and modifies an ISAM CCW string, executable.

Module Name	Entry Points	Attributes, Function
DMKIUA		Resident, executable.
	DMKIUACP	Prepare to handle CP IUCV request on behalf of the system, executable.
	DMKIUACU	Prepare to handle CP IUCV request on behalf of a virtual machine, executable.
	DMKIUAEP	Prepare to handle user IUCV request, executable.
	DMKIUAQU	IUCV message queue routine, executable.
	DMKIUAPD	Validate an IUCV communications path, executable.
	DMKIUAPL	Validate and synchronize communications on an IUCV system path, executable.
DMKIUB		Resident, executable.
	DMKIUBRK	Reflect IUCV messages and replies to a virtual machine, executable.
	DMKIUBTB	Table of IUCV system service entry points, executable.
DMKIUC		Pageable, executable.
	DMKIUCEP	Processes requests for IUCV functions (Accept and Connect), executable.
DMKIUE		Resident, executable.
	DMKIUEEP	Processes requests for IUCV function (Receive), executable.
	DMKIUELR	Check logical record lengths for APPC/VM, executable.
	DMKIUERC	Handles data movement for APPC/VM RECEIVE, executable.
DMKIUG		Pageable, executable.
	DMKIUGEP	Processes requests for IUCV functions (Reject and Purge), executable.
	DMKIUGGP	IUCV routine to obtain a communications path, executable.
DMKIUJ		Pageable, executable.
	DMKIUJEP	Processes requests for IUCV function (Sever), executable.
DMKIUL		Pageable, executable.
	DMKIULEP	Processes requests for IUCV functions (Reply and Test Completion), executable.
	DMKIULRP	Handles data movement for APPC/VM SENDs when an answer area is predefined, executable.
DMKIUN		Resident, executable.
	DMKIUNEP	Process requests for IUCV functions (Describe, Send, Set Mask, Set Control Mask, and Test Message), executable.
	DMKIUNIN	IUCV external interrupt queueing routine, executable.
DMKIUP		Pageable, executable.
	DMKIUPEP	Process requests for IUCV functions (Declare Buffer, Quiesce, Query, Resume, and Retrieve Buffer), executable.
DMKIUS		Resident, executable.
	DMKIUSEP	Handles APPC/VM functions (Send Data, Send Confirm, Send Confirmed, Send Error, Send Request, and Receive), executable.
	DMKIUSET	Sets path states for APPC/VM functions, executable.

Module Name	Entry Points	Attributes, Function
DMKJRL		Pageable, executable.
	DMKJRLIL	Processes LINKs with invalid passwords, executable.
	DMKJRLLO	Processes LOGONs with invalid passwords, executable.
	DMKJRLQU	Processes the QUERY command, executable.
	DMKJRLSC	Processes LINKs which are successful, executable.
	DMKJRLSE	Processes the SET JOURNAL command, executable.
DMKLD00E		Loader - utility program.
	LDRGEN	Loads assembled program modules into storage at locations other than those assigned by the assembler. It completes linkage among the modules and transfers control to one of the loaded modules for execution.
DMKLNK		Pageable, executable.
	DMKLNKIN	Links to a virtual DASD because of an issued LINK command, executable.
	DMKLNKSB	Handles the LINK directory statement at logon time, executable.
	DMKLNKSS	LINK command handler after an MSS mount, executable.
DMKLNM		Pageable, executable.
	DMKLNMSG	Processes error and response messages for the LINK command, executable.
DMKLOC		Pageable, executable.
	DMKLOCK	Allows a system resource to be marked in use or not available by a unique 8-character name, executable.
	DMKLOCKD	Dequeues a locked name, executable.
	DMKLOCKQ	Queues or locks a name, executable.
	DMKLOCKT	Tests to determine if a name is locked, executable.
DMKLOG		Pageable, executable.
	DMKLOGB	Processes the AUTOLOG command, executable.
	DMKLOGON	Checks syntax for the LOGON and AUTOLOG commands, and verifies the given userid and password, executable.
	DMKLOGOP	Logs on the operator, executable.
DMKLOH		Pageable, executable.
	DMKLOHRC	Updates VMBLOK to LOGON a user or to RECONNECT a user, executable.
DMKLOJ		Pageable, executable.
	DMKLOJEP	Creates virtual devices for the virtual machine being logged on, executable.

Module Name	Entry Points	Attributes, Function
DMKLOK		Resident, executable. Handles all locking requests when CP is in attached processor mode.
	DMKLOKCT	Counters for monitor, non-executable.
	DMKLOKDF	Processes an obtain, defer lock request, executable.
	DMKLOKDS	Dispatcher queues lockword, non-executable.
	DMKLOKFR	DMKFRE lockword, non-executable.
	DMKLOKND	Point beyond last lockword, executable.
	DMKLOKIO	I/O lockword, non-executable.
	DMKLOKPS	Processor spin lock requests, executable.
	DMKLOKRL	RUNLIST lockword, non-executable.
	DMKLOKRM	Real storage management lockword, non-executable.
	DMKLOKSI	Spin indicator lockword, non-executable.
	DMKLOKSP	Processes an obtain request for a spin lock that previously failed, executable.
	DMKLOKSW	Process an obtain and release request for VMBLOK requested by SWTCHMV macro, executable.
	DMKLOKSY	System lockword, non-executable.
	DMKLOKTR	Timer request lockword, non-executable.
	DMKLOKVM	Processes an obtain, defer request for VMBLOK lock, executable.
DMKLOM		Pageable, executable.
	DMKLOMSG	Constructs and sends logon-related messages to a user or to the operator, executable.
	DMKLOMSS	Handles the allocation of a MSS disk after the volume has been mounted, executable.
DMKMCC		Pageable, executable.
	DMKMCCCL	Handles first level MONITOR command processing, executable.
DMKMCD		Pageable, executable.
	DMKMCDIN	Processes MONITOR INTERVAL commands, executable.
	DMKMCDLI	Processes MONITOR LIMIT commands, executable.
	DMKMCDTI	Processes MONITOR TIME commands, executable.
	DMKMCDSE	Processes MONITOR SEEKS commands, executable.
	DMKMCDST	Processes MONITOR STOP commands, executable.
DMKMCH		Resident, executable.
	DMKMCHIN	Processes a machine check interruption, executable.
	DMKMCHLM	Counts field for soft errors, executable.
	DMKMCHSE	Continue processing soft errors with system lock, executable.
	DMKMCHST	Terminate the system, executable.
	DMKMCHTR	Entered via TRQBLOK to enable soft error recording, executable.
DMKMCI		Pageable, executable.
	DMKCIMS	Enables or disables soft machine check recording, executable.

Module Name	Entry Points	Attributes, Function
DMKMCT		Resident, executable. This module is called by the machine check handler in attached processor mode.
	DMKMCTAF	Executable.
	DMKMCTFS	Handles unsuccessful SIGP recovery, executable.
	DMKMCTFL	DMKMCT flags, non-executable.
	DMKMCTMA	Handles malfunction alert, executable.
	DMKMCTPF	Address of the PSA page of a terminated processor, executable.
	DMKMCTPR	Handles processor recovery, executable.
	DMKMCTPT	Handles processor termination, executable.
	DMKMCTST	Handles system termination, executable.
	DMKMCTVM	Terminates virtual user address, executable.
DMKMES		Pageable, non-executable. This is the CP message repository. It contains most CP messages. Module DMKERM references this message repository to write out messages.
DMKMHC		Resident, executable.
	DMKMHCCP	Handles MSSF service requests for CP, executable.
	DMKMHCIN	Processes MSSFCALL external interrupts, executable.
	DMKMHCLK	IOCP write lockword, non-executable.
	DMKMHCRE	Releases a virtual machine HCBLOK at logoff, executable.
	DMKMHCVM	Handles MSSF service requests for a virtual machine, executable.
DMKMHV		Pageable, executable.
	DMKMHVSM	Simulates DIAGNOSE X'0080' (MSSFCALL) for a virtual machine, executable.
DMKMIA		Pageable, executable. Provides various facilities associated with automatic monitoring using spool files.
	DMKMIACC	Used for MONITOR CLOSE processing, executable.
	DMKMIADL	Used for DMKMCC display function, executable.
	DMKMIAEN	Used to invoke a MONITOR STOP command, executable.
	DMKMIAIN	Used to invoke a MONITOR START command, executable.
	DMKMIAKC	Used to invoke a MONITOR CLOSE command, executable.
	DMKMIAMU	Generates informational messages for monitor user, executable.
	DMKMIARO	Opens monitor spool file, gets SFB, etc., executable.
	DMKMIAWO	Writes a monitor data buffer to a spool file buffer, executable.
DMKMID		Pageable, executable.
	DMKMIDNT	Changes the date in the system low storage at midnight and resets the clock comparator for the next midnight occurrence. DMKMID also sends messages to all users about the date change, executable.
DMKMNI		Pageable, executable.
	DMKMNIDK	Constructs spool file header record, executable.
	DMKMNIFI	Completes monitor shutdown, executable.
	DMKMNISH	Initializes MONITOR shutdown, executable.
	DMKMNIST	Monitors auto start/stop processing, executable.
	DMKMNITH	Handles monitor tape header processing, executable.
	DMKMNITR	Writes the MONITOR trailer record, executable.

Module Name	Entry Points	Attributes, Function
DMKMNJ		Pageable, executable.
	DMKMNJDS	Displays automatic monitoring information defined by SYSMON macro in DMKSYS, executable.
	DMKMNJGT	Initializes the monitor TRQBLOK for timer driven event sampling, executable.
	DMKMNJSP	Handles monitor processing for SPOOL to USERID parameters of START command, executable.
DMKMNT		Pageable, executable.
	DMKMNTIO	Marks the I/O devices online during system initialization, executable.
DMKMON		Pageable, executable. Processes commands and requests associated with the MONITOR, including MONITOR CALL interruptions within CP, executable.
	DMKMONIO	Processes tape interruptions returned by DMKIOS, executable.
	DMKMONMI	Processes a MONITOR CALL program interruption, executable.
	DMKMONPR	Gets space for monitor record and manages buffers, executable.
	DMKMON00	Handles PERFORM (class zero) data collection routine, executable.
DMKMOO		Pageable, executable. Paged in and locked when MONITOR START command is issued.
	DMKMOOTI	Handles timer request interruptions, executable.
	DMKMOOX1	Number of paging SIO's, non-executable.
	DMKMOOX2	Number of user's in Q2, non-executable.
	DMKMOOY1	Number of real external interrupts, non-executable.
	DMKMOOY2	Number of resume second procedures issued, non-executable.
	DMKMOO40	Handles USER (class four) data collection routine, executable.
DMKMSG		Pageable, executable. Transmits messages to logged-on users for the MESSAGE, SMSG, or WARNING commands. Receives and retransmits lines for the ECHO command for the number of times specified.
	DMKMSGAL	*SPL 'WNG' informational 'MSG' processor, executable.
	DMKMSGCN	Incoming IUCV connections, executable.
	DMKMSGC2	Incoming IUCV connections to *MSGALL, executable.
	DMKMSGEC	ECHO command processor, executable.
	DMKMSGIU	Entry point for message via IUCV, executable.
	DMKMSGMA	Processes the MESSAGE and MESSAGE ALL commands for primary system operators and system resource operators, executable.
	DMKMSGMG	Processes the MESSAGE command for general users, executable.
	DMKMSGMS	*SPL 'WNG' processor, executable.
	DMKMSGM5	SPL informational message, executable.
	DMKMSGNH	MSGNOH command processor, executable.
	DMKMSGSM	SMSG command processor, executable.
	DMKMSGSV	IUCV connection severed, executable.
	DMKMSGS2	*MSGALL path connection severed, executable.
	DMKMSGWN	WARNING command processor, executable.

Module Name	Entry Points	Attributes, Function
DMKMSW		Resident, executable.
	DMKMSWR	Allows system communication with the operator for the enhancement of error recovery procedures, executable.
DMKNEA		Pageable, executable.
	DMKNEAAH	Processes the network ATTACH function, executable.
	DMKNEADF	Accesses the RDEVBLOK and NICBLOK, executable.
	DMKNEADH	Processes the network DETACH function, executable.
	DMKNEARV	Accesses the RDEVBLOK, executable.
	DMKNEAVR	Accesses the RDEVBLOK and the NICBLOK, executable.
DMKNEM		Pageable, executable.
	DMKNEMOP	Gets a 5-byte mnemonic opcode for a System/370 binary opcode, executable.
DMKNES		Pageable, executable. Processes NETWORK operands as follows: POLLDLAY SHUTDOWN DISPLAY VARY TRACE
	DMKNESDS	Processes the NETWORK DISPLAY command, executable.
	DMKNESEP	Processes the NETWORK VARY EP command to switch an NCP communication line to EP mode, executable.
	DMKNESHD	Processes the NETWORK SHUTDOWN command, executable.
	DMKNESPL	Processes the NETWORK POLLDLAY command, executable.
	DMKNESTR	Processes the NETWORK TRACE command, executable.
	DMKNESWN	Processes the NETWORK VARY NCP command to switch an EP communication line to NCP mode, executable.
DMKNET		Pageable, executable. Decodes NETWORK command and enables bisync lines.
	DMKNETAE	Enable binary synchronous lines and remote stations, executable.
	DMKNETDB	Processes the NETWORK DISABLE command, executable.
	DMKNETEN	Processes the NETWORK ENABLE command, executable.
	DMKNETQU	Processes the NETWORK QUERY command, executable.
	DMKNETVA	Processes the NETWORK VARY command, executable.
DMKNLD		Pageable, executable.
	DMKNLDR	Loads the 3705 network control program. These routines may be called by a console command from DMKNET or internally by DMKCPI (for LOAD) or DMKRNH (for DUMP), executable.
DMKNLE		Pageable, executable.
	DMKNLEMP	Dumps the 3705 Network Control Program, executable.
DMKNMT		Pageable, executable.
	DMKNMTBL	Constructs an Image Library for TEXT files on user disks, executable.

Module Name	Entry Points	Attributes, Function
DMKOPE		Pageable, executable.
	DMKOPEAC	Sets up auto monitor and logs on the AUTOLOG1 user's virtual machine during system initialization, executable.
	DMKOPEDC	Disconnects the system operator during system initialization, executable.
	DMKOPEEM	Returns from DMKCNS for entry DMKOPERC, executable.
	DMKOPELO	Logs on the system operator during system initialization, executable.
	DMKOPERC	Locates the operator's console during system initialization, executable.
DMKOPR		Resident, executable.
	DMKOPRWT	Provides the necessary support for the VM/SP system console. Certain routines within the control program cannot call DMKQCN to issue writes to the system console. This module determines the systems primary console and builds a channel program to handle the requested call, executable.
DMKOVR		Standalone, executable. Uses preallocated cylinders to build a Command Class Override file on a specified DASD volume.
	DMKOVRDE	This is the override entry point, executable.
DMKPAG		Resident, executable.
	DMKPAGCC	Count of calls to this routine, non-executable.
	DMKPAGDP	Flag for setting PCI on all 2305 requests, non-executable.
	DMKPAGEX	Anchor of available EXT-CKD paging I/O blocks, non-executable.
	DMKPAGHI	"DMKFREHI," non-executable.
	DMKPAGIC	Number of paging SIO's over which to measure load, non-executable.
	DMKPAGIO	Constructs IOBLOKs and schedules the tasks that move virtual storage pages between auxiliary storage and main storage. It also calculates the total system paging paging load at user-specified intervals, executable.
	DMKPAGLO	"DMKFRELO" minus length of one paging IOBLOK, non-executable.
	DMKPAGPS	Count of paging SIO's, non-executable.
	DMKPAGQ	Anchor for list of CPEXBLOKS for in-transit pages, non-executable.
	DMKPAGQR	Maintain page I/O for query paging command, non-executable.
	DMKPAGSK	Anchor of available page CKD/FBA I/O blocks, non-executable.
	DMKPAGSP	End of page load measurement period, non-executable.
	DMKPAGST	Start of page load measurement period, non-executable.
DMKPAH		Resident, executable.
	DMKPAHIO	Handles paging interruptions, executable.
DMKPEI		Pageable, executable.
	DMKPEINT	Creates a PEXBLOK chain for PER command, executable.
DMKPEL		Pageable, executable.
	DMKPELCH	Changes/Merges two PEXBLOK chains, executable.
	DMKPELCR	Computes PER control registers 9, 10, and 11, executable.
	DMKPELSD	Completes handling of the CP PER command, executable.

Module Name	Entry Points	Attributes, Function
DMKPEN		Pageable, executable.
	DKMPEND	Handles the PER END subcommand, executable.
	DMKPENDA	Ends all CP PER tracing activity, executable.
	DMKPENGT	Handles the PER GET subcommand, executable.
	DMKPENSV	Handles the PER SAVE subcommand, executable.
DMKPEQ		Pageable, executable.
	DMKPEQRY	Handles the QUERY PER command, executable.
DMKPER		Pageable, executable.
	DMKPERIL	Handles PER program interrupts, executable.
DMKPET		Pageable, executable.
	DMKPETAB	Produces display of TRACEBACK table, executable.
	DMKPETAL	Produces instruction display for PER event, executable.
DMKPGM		Pageable, executable.
	DMKPGMEP	Invoked by the dispatcher if the dispatcher has a CPEXBLOK stacked for page migration. It migrates pages from preferred backing storage to nonpreferred backing storage. When page migration is completed, it checks to see whether the swap table migration flag is set. If it is, it stacks a CPEXBLOK in DMKSTKCP for swap table migration, entry point DMKSTRSM, executable.
	DMKPGMST	Pointer to counters, non-executable.
	DMKPGMUS	Invoked if page migration or swap table migration is invoked by the MIGRATE command. If the command is valid, a CPEXBLOK is stacked with an entry point of DMKPGMX to migrate all users, or an entry point of RESETUS to migrate one user, executable.
DMKPGS		Resident, executable.
	DMKPGSPO	Releases all the pages of a user's virtual storage from the real storage and from auxiliary storage on the paging device, executable.
	DMKPGSPP	Releases a specified part of virtual storage, executable.
	DMKPGSPR	Calls DMKPTRPU to ensure that the user is not in page wait and then releases the address range contained in R1 through R2. It also unlocks any pages that might have been locked, executable.
	DMKPGSPS	Releases a named system from user's virtual storage, executable.
	DMKPGSSS	Releases virtual storage but bypass any virtual storage which contains a named system, executable.

Module Name	Entry Points	Attributes, Function
DMKPGT		Resident, executable. DASD storage management.
	DMKPGTAE	End of list pointers for FBA temporary disk generic anchors, non-executable.
	DMKPGTAN	Temporary disk - 3370, non-executable.
	DMKPGTAT	Pointers to FBA temporary disk generic anchors for 3310, non-executable.
	DMKPGTAW	Temporary disk - 3310, non-executable.
	DMKPGTA0	Temporary disk - 3350, non-executable.
	DMKPGTA4	Temporary disk - 2314, non-executable.
	DMKPGTA5	Temporary disk - 2305, non-executable.
	DMKPGTCG	Allocates contiguous space for a 3704/3705 dump, executable.
	DMKPGTDF	Drum limit flags and count, non-executable.
	DMKPGTDG	Allocates contiguous DASD space for a CP dump reader file that is being off Loaded by CPs SPTAPE LOAD command, executable.
	DMKPGTDK	Specifies the total number of preferred Moveable Head cylinders allocated for paging (DISKPAG), executable.
	DMKPGTDM	Specifies the total number of preferred Fixed Head cylinders allocated for paging (DRUMPAG), executable.
	DMKPGTDS	Number of pages for system dump, non-executable.
	DMKPGTDT	Allocates contiguous space for the system dump file from DUMP or TEMP DASD space, executable.
	DMKPGTFC	Count of FH index entries, non-executable.
	DMKPGTFE	End of list pointers for FBA preferred anchors, non-executable.
	DMKPGTFI	FH index entries (max = 5), non-executable.
	DMKPGTFP	Pointers to FBA preferred anchors for 3370, non-executable.
	DMKPGTFT	Pointers to FBA general anchors for 3370, non-executable.
	DMKPGTF0	Preferred FH paging - 3330, non-executable.
	DMKPGTF5	Preferred FH paging - 2305, non-executable.
	DMKPGTMC	Count of MH index entries, non-executable.
	DMKPGTMI	MH index entries (max = 8), non-executable.
	DMKPGTMN	Preferred MH paging - 3370, non-executable.
	DMKPGTMS	Message switch, non-executable.
	DMKPGTMW	Preferred MH paging - 3310, non-executable.
	DMKPGTM0	Preferred MH paging - 3330, non-executable.
	DMKPGTM4	Preferred MH paging - 2314, non-executable.
	DMKPGTPC	Specifies the number of preferred Moveable Head pages allocated, non-executable.
	DMKPGTPG	Allocates a page of DASD storage for either virtual storage paging or for spool file page buffers, executable.
	DMKPGTPL	Specifies the limit of preferred Moveable Head pages for use in setting MHFULL flag on, non-executable.
	DMKPGTPM	Allocate one DASD page of non-preferred for page migration, executable.
	DMKPGTPN	Specifies the percentage limit set by the SET SRM MHFULL command, non-executable.

Module Name	Entry Points	Attributes, Function
DMKPGT (cont)		Resident, executable.
		DASD storage management.
	DMKPGTSG	Allocates a page of DASD storage for spooling.
	DMKPGTTC	Count of temporary index entries, non-executable.
	DMKPGTTI	Temporary index entries (max = 9), non-executable.
	DMKPGTTM	Maximum number of pages available, non-executable.
	DMKPGTTN	General spooling/paging - 3370, non-executable.
	DMKPGTTU	Number of temporary pages in use, non-executable.
	DMKPGTTW	General spooling/paging - 3310, non-executable.
	DMKPGTT0	General spooling/paging - 3330, non-executable.
	DMKPGTT4	General spooling/paging - 2314, non-executable.
	DMKPGTT5	General spooling/paging - 2305, non-executable.
	DMKPGT4A	Temporary disk - 3310, non-executable.
	DMKPGT4F	Preferred FH paging - 3340, non-executable.
	DMKPGT4M	Preferred MH paging - 3340, non-executable.
	DMKPGT4T	General spooling/paging - 3340, non-executable.
	DMKPGT5A	Temporary disk - 3350, non-executable.
	DMKPGT5F	Preferred FH paging - 3350, non-executable.
	DMKPGT5M	Preferred MH paging - 3350, non-executable.
	DMKPGT5T	General spooling/paging - 3350, non-executable.
	DMKPGT7A	Temporary disk - 3375, non-executable.
	DMKPGT7M	Preferred MH paging - 3375, non-executable.
	DMKPGT7T	General spooling/paging - 3375, non-executable.
	DMKPGT8A	Temporary disk - 3380, non-executable.
	DMKPGT8F	Preferred FH paging - 3380, non-executable.
	DMKPGT8M	Preferred MH paging - 3380, non-executable.
	DMKPGT8T	General spooling/paging - 3380, non-executable.
	DMKPGT90	90% of available pages, non-executable.
DMKPGU		Resident, executable.
		Performs DASD storage management.
	DMKPGUBN	Number of virtual buffers/8, executable.
	DMKPGUDG	Builds RECBLOK chain for FBA device dump files, executable.
	DMKPGUDT	Builds RECBLOK chain for FBA device dump files, executable.
	DMKPGUDU	Deallocates contiguous DASD space for system dump, executable.
	DMKPGUPR	Releases DASD storage used for virtual storage paging, executable.
	DMKPGUSD	Releases one page of DASD storage used for virtual spooling, executable.
	DMKPGUSP	Releases one page of DASD storage used for virtual paging, executable.
	DMKPGUSR	Releases a set of DASD pages that belong to a spool file that is no longer needed, executable.
	DMKPGUVG	Allocates a page of virtual storage belonging to the CP paging VMBLOK, executable.
	DMKPGUVR	Releases a virtual storage page, executable.
DMKPIA		Pageable, non-executable.
	DMKPIALD	Contains the FOB buffer images that the LOADBUF command uses to load the Font Offset Buffer of the 3289 Model 4 printer, non-executable.

Module Name	Entry Points	Attributes, Function
DMKPIB		Pageable, non-executable.
	DMKPIBLD	Contains the buffer images that the LOADBUF command uses to load the UCSB of the IBM 3262 Printer, Models 1 and 11, non-executable.
DMKPRG		Resident, executable.
	DMKPRGCT	Count of user program interrupts reflected to users, executable.
	DMKPRGC8	CP 'MONITOR' storage, used by "MONITOR START," non-executable.
	DMKPRGIN	Processes a hardware program interruption, executable.
	DMKPRGMC	CP 'MONITOR' communication pointer, filled in by "START," non-executable.
	DMKPRGMI	CP 'MONITOR' entry pointer, filled in by "MONITOR START," non-executable.
	DMKPRGMO	Address of DMKMOO program interrupt handler, non-executable.
	DMKPRGRF	Reflects an SVC interruption to the virtual machine, executable.
	DMKPRGSM	Simulates a virtual program interruption, executable.
	DMKPRGTI	CP 'MONITOR' timer interval, used by DMKMON, non-executable.
DMKPRV		Resident, executable.
	DMKPRVLG	Simulates a privileged operation, executable.
DMKPRW		Resident, executable.
	DMKPRWIP	Simulates IPTE instruction, executable.
	DMKPRWTB	Simulates TEST BLOCK instruction, executable.
	DMKPRWTP	Simulates TPROT instruction, executable.
DMKPSA		Resident, executable.
	DMKPSACC	Checks a shared page for change, executable.
	DMKPSADU	PSW restart processing. Forces an SVC 0 type of dump, executable.
	DMKPSAER	Registers saved here for external interrupts, non-executable.
	DMKPSAFC	Checks fetch protection per the CAW key, executable.
	DMKPSAFP	Checks for fetch protection violation per PSW key, executable.
	DMKPSAID	Gets virtual address for any instruction, executable.
	DMKPSANX	Number of external interrupts, non-executable.
	DMKPSARR	Gets the virtual address for an RR instruction, executable.
	DMKPSARS	Gets the virtual address for RS, SI, or SS instruction, executable.
	DMKPSARX	Gets the virtual address for an RX instruction, executable.
	DMKPSASC	Checks storage protection per the CAW key, executable.
	DMKPSASP	Checks for a storage protection violation per the PSW key, executable.
	DMKPSAST	Address of start of trace table (TRACSTRT), non-executable.

Module Name	Entry Points	Attributes, Function
DMKPTR		Resident, executable. Manages the inventory of real system pages, provides real storage space for CP functions and for pages of user and CP virtual storage.
	DMKPTRAN	Translates user virtual storage address to a real storage address, executable.
	DMKPTRCP	Table of indices for cortable searching, non-executable.
	DMKPTRCS	Number of times entire ULIST examined, non-executable.
	DMKPTRCT	Number of calls to DMKPTRAN, non-executable.
	DMKPTRFA	Block of storage used for extending the system, non-executable.
	DMKPTRFC	Number of requests to obtain a page frame, non-executable.
	DMKPTRFD	Place a page frame on the FREELIST after it has been written out to DASD, executable.
	DMKPTRFE	Extend processing page selection, executable
	DMKPTRFF	Number of pages taken from flushed page list, non-executable.
	DMKPTRFN	Number of free pages currently available - initialized by DMKCPINT, non-executable.
	DMKPTRFP	Resume extend processing once a page frame has been obtained, executable.
	DMKPTRFQ	Queue of deferred requests for free core, non-executable.
	DMKPTRFR	Gets a page of real storage, executable.
	DMKPTRFT	Releases a page of real storage, executable.
	DMKPTRF0	Number of times DMKPTRFR was called and no pages were available, non-executable.
	DMKPTRF1	Pointer to the cortable entry for the first free page - initialized by DMKCPINT, non-executable.
	DMKPTRLK	Locks a page of real storage and processes the CP assist instruction, PTRLK (E602), executable.
	DMKPTRLX	Microcode assist to lock a page frame. Referenced by DMKCPI, non-executable.
	DMKPTRPL	Anchor used by CP assist for unlocking pages, non-executable.
	DMKPTRPR	Pages reclaimed, non-executable.
	DMKPTRPW	Called to defer execution of system reset functions when users virtual machine is in page wait, executable.
	DMKPTRRC	Current number of resident reserved pages, non-executable.
	DMKPTRRF	Number of pages looked at during STEAL, non-executable.
	DMKPTRRL	High water limit for reserved page user, non-executable.
	DMKPTRRM	Storage size (set by DMKCPI), non-executable.
	DMKPTRRQ	Page read request queue, non-executable.
	DMKPTRRS	Reset pages belonging to user, executable.
	DMKPTRRU	Address of VMBLOK for current reserved page user, non-executable.
	DMKPTRSC	Number of resident, system shared pages, non-executable.
	DMKPTRSN	Page selection pointer, non-executable.
	DMKPTRSS	Number of pages stolen from active users, non-executable.
	DMKPTRSW	Number of pages being swapped, non-executable.
	DMKPTRUL	Unlocks a page of real storage and processes the CP assist instruction, PTRUL (E603), executable.
	DMKPTRUX	Microcode assist for unlocking a page. Referenced by DMKCPI, non-executable.
	DMKPTRU1	Address of first cortable entry on flush list, non-executable.
	DMKPTRWQ	Page write request queue, non-executable.
	DMKPTRXX	Lock to determine if user changed page, non-executable.

Module Name	Entry Points	Attributes, Function
DMKQCN		Resident, executable.
	DMKQCNFT	Tells DMKFRE where this fret is invoked, non-executable.
	DMKQCNPL	Initiates console write requests where the calling module has built the parameter, executable.
	DMKQCNWT	Starts and queues a console write request. If a larger than 4K buffer is required, an indirect address list is used to address the noncontiguous buffer, executable.
DMKQCO		Resident, executable.
	DMKQCOCL	Clears CONTASK stack and returns all blocks to free storage, executable.
	DMKQCOCS	Console spooling validation, executable.
	DMKQCOET	Processes completed CONTASKs for virtual console spooling return or no return options, and returns the CONTASK blocks to free storage, executable.
	DMKQCOFT	Inform DMKFRE where this request is from, non-executable.
	DMKQCONQ	Queues up CONTASK, executable.
	DMKQCOPL	Queue console read request for modules with pre-built call parameter lists, executable.
	DMKQCORD	Starts and queues a console Read request, executable.
	DMKQCOSY	Synchronizes virtual machine console activity with internal supervisor activity, executable. (used during virtual system reset and logoff)
DMKQCP		Pageable, executable.
	DMKQCPTO	Disconnects the virtual machine. Sets TOD clock comparator request to logoff the virtual machine after a 15 minute time delay, executable.
DMKQCQ		Resident, executable.
	DMKQCQED	Edits data stream for console writes, executable.
DMKQVM		Resident, executable.
	DMKQVMCP	Address of CP page 0, non-executable.
	DMKQVMCU	Entry to reflect cue to V=R machine, executable.
	DMKQVMEP	Process the CP QVM command, executable.
	DMKQVMRS	Performs the switch back to VM/370 for non-370E operating system, executable.
	DMKQVMRT	Performs the switch to native mode for the V=R user, executable.
	DMKQVMRX	Performs the switch back to VM/370 for 370E operating system, executable.
	DMKQVMTS	Time stamp for end of IOS QVM interrupt window, non-executable.
	DMKQVMVP	Saved restart PSW for V=R virtual machine, non-executable.
DMKREI		Pageable, executable. Handles re-IPL processing for protected application environment users.
	DMKREIPL	Re-IPL protected application environment users, executable.
DMKRET		Resident, executable.
	DMKRETGT	Get a line from the retrieve buffer, executable.
	DMKRETPT	Puts an input line into the retrieve buffer, executable.

Module Name	Entry Points	Attributes, Function
DMKRGA		Resident, executable.
	DMKRGACC	Executable.
	DMKRGAIN	This is the second-level interruption handler for remote 3270 stations. This module supports the 3270 remote display and printer stations. It processes interruptions and CCWs for the remote stations, including message handling and screen management, executable.
	DMKRGATM	Processes time interrupts for the following 3270 display conditions: -completion of poll delay period -completion of 60 second delay on priority -messages -more timeout -second Not Accept message. Executable.
	DMKRGASP	Complete processing full screen WSF and QUERY, executable.
	DMKRGA2	If a larger than 4K buffer is required, an indirect address list is used to address the noncontiguous buffer, executable.
DMKRGB		Resident, executable. Supports the 3270 remote display and printer stations. It processes interruptions and CCWS for the stations including message handling and screen management.
	DMKRGBCL	Clear the display screen after full screen I/O, executable.
	DMKRGBCO	Continue output operations on a currently selected remote station, executable.
	DMKRGBEN	Enables and disables bisync lines and remote stations, executable.
	DMKRGBFM	Address of CONTASK/LOGO buffer and logo size as initialized by DMKGRTFM, non-executable.
	DMKRGBIC	Initializes and schedules CONTASKs. If a larger than 4K buffer is required, an indirect address list is used to address the noncontiguous buffer, executable.
	DMKRGBMT	Formats the display screen, executable.
	DMKRGBPL	Issue a specific or general poll, executable.
	DMKRGBRE	Starts I/O on a teleprocessing line, executable.
	DMKRGBSL	Select a remote station, executable.
	DMKRGBSN	Screens NICBLOK list for output messages; do general poll if none found, executable.
	DMKRGBUP	Dates a remote 3270 screen, executable.
DMKRGC		Resident, executable. Processes 3270 input data and status messages after validation by DMKRGA.

Module Name	**Entry Points**	**Attributes, Function**
DMKRGD		Resident, executable.
	DMKRGDOB	Performs the blocking of output data to display terminals and updates the CONTASK. If a larger than 4K buffer is required, an indirect address list is used to address the noncontiguous buffer, executable.
	DMKRGDOI	Performs the blocking of output data to display terminals and updates the CONTASK. If a larger than 4K buffer is required, an indirect address list is used to address the noncontiguous buffer, executable.
DMKRGE		Resident, executable.
	DMKRGEIO	Stack I/O interrupt to DMKVIOIN, executable.
	DMKRGESK	Handles skip processing for BSC and VM/SNA devices, executable.
DMKRIO		Resident, non-executable. Exists as a CSECT and defines the machines configuration. A basic DMKRIO is shipped with the system. DMKRIO can be changed at system generation or whenever new machines are added by using the appropriate macros.
DMKRND		Residency not applicable. Invoked via the NCPDUMP command in CMS, executable. This is the interface between the VM/SP dump spool file and the OS-SSP dump format program for printing and formatting dumps of the 3704 and 3705 communications controllers.
	NCPDUMP	370X dump processor interface, executable.
DMKRNH		Resident, executable.
	DMKRNHCT	Statistical counter block, non-executable.
	DMKRNHIC	Initializes and schedules the CONTASK fields that comprise the 37xx Network Control Program transmission header, executable.
	DMKRNHIN	This is the secondary interruption handler for the 37xx communication controllers, executable. It is read when operating in NCP or PEP mode.
	DMKRNHND	Schedules control functions for the 37xx Network Control Program, executable.
	DMKRNHTG	Request tag source field, executable.
	DMKRNHTR	Userid for BTU trace control, non-executable.
DMKRPA		Resident, executable. Virtual storage mapping.
	DMKRPAGT	Page-in from DASD to users virtual storage, executable.
	DMKRPAPT	Page-out to DASD from users virtual storage, executable.
DMKRPD		Pageable, executable.
	DMKRPDEP	Security DIAGNOSE Routine, executable.

Module Name	Entry Points	Attributes, Function
DMKRPI		Pageable, executable.
		Access Verification Routines.
	DMKRPICN	CONNECT Request, executable.
	DMKRPIIL	MESSAGE Request, executable.
	DMKRPIQS	QUIESCE Request, executable.
	DMKRPIRA	LINK Request, executable.
	DMKRPIRM	RESUME Request, executable.
	DMKRPISV	SEVER Request, executable.
DMKRPW		Pageable, executable.
	DMKRPWEP	Logon password verification routine, executable.
DMKRSE		Pageable, executable.
		Real UR device I/O error handler.
	DMKRSERR	Retries and attempts to recover from real unit record device I/O errors, executable.
DMKRSF		Pageable, executable.
		Real UR device I/O error handler.
	DMKRSFPB	Purges 3800 page buffer on a 3800 printer when called due to an error, executable.
	DMKRSFPR	Gets 3211 type printers error information, executable.
	DMKRSFSD	Formats 3800 printer hardware environmental counters, executable.

Module Name	Entry Points	Attributes, Function
DMKRSP		Resident, executable. Manages all spooling operations on the real system unit record devices. This includes printing and punching user-created spool files and reading and queueing reader files from the real card reader.
	DMKRSPAC	The anchor for all accounting card buffers that are waiting to be punched, non-executable.
	DMKRSPCV	Time of day clock value used by DMKCKP, non-executable.
	DMKRSPDC	Device code for checkpoint cycles, non-executable.
	DMKRSPDE	Displacement to end of RDEVs in slot map, non-executable.
	DMKRSPDL	The anchor for spool file delete chain, non-executable.
	DMKRSPDP	Non-executable.
	DMKRSPEC	Number of entries available for checkpointing, non-executable.
	DMKRSPER	Processes spooling errors (ERP), executable.
	DMKRSPEX	Processes spooling operations, executable. Entered via a GOTO when DMKDSPCH unstacks an IOBLOK with an interruption for the spooling unit record device.
	DMKRSPHQ	The anchor for system hold queue, non-executable.
	DMKRSPID	Gets a unique spoolid, non-executable.
	DMKRSPLP	The anchor for logical printer device block, executable.
	DMKRSPML	Checkpoint map address list, non-executable.
	DMKRSPMN	Active monitor SFBLOK address, non-executable.
	DMKRSPM1	Virtual address of checkpoint map 1, non-executable.
	DMKRSPM2	Virtual address of checkpoint map 2, non-executable.
	DMKRSPM3	Virtual address of checkpoint map 3, non-executable.
	DMKRSPM4	Virtual address of checkpoint map 4, non-executable.
	DMKRSPM5	Virtual address of checkpoint map 5, non-executable.
	DMKRSPM6	Virtual address of checkpoint map 6, non-executable.
	DMKRSPND	End checkpoint cycle, non-executable.
	DMKRSPNM	Anchor for pointer to next slot map, non-executable.
	DMKRSPPC	Pages/cycles for IPL device, non-executable.
	DMKRSPPR	The anchor for printer file chain, non-executable.
	DMKRSPPU	The anchor for the punch file chain, non-executable.
	DMKRSPRD	The anchor for the chain of SFBLOKS for all reader files waiting to be processed, non-executable.
	DMKRSPSC	Active spool file counter, non-executable.
	DMKRSPSM	Virtual address of spool fileid bit map, non-executable.
	DMKRSPSP	The anchor for spool tape delete chain, non-executable.
	DMKRSPSR	Real address of spool fileid bit map, non-executable.
	DMKRSPST	Checkpoint start cycle, non-executable.
	DMKRSPSW	Switch for checkpointing, non-executable.
	DMKRSPTP	SYSRES device class (RDEVTYPC), non-executable.
	DMKRSPTR	Non-executable.
	DMKRSPWA	Warm start device information, non-executable.
	DMKRSP83	Reset CCW for 3811 printer control unit, executable.
	DMKRSP9P	90% of maximum slots, non-executable.
	DMKRSP9R	90% message grace value, non-executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKRSQ		Pageable, executable. Handles the spool file buffers when data chaining between DASD buffer is required. For example, for 3800 printer Load Graphic Modification or Load Copy Modification CCW.
	DMKRSQDC	Obtains buffers needed for data chaining and after the associated CCWs, executable.
	DMKRSQFR	Free the buffers obtained by DMKRSQDC, executable.
DMKRST		Pageable, executable. Handles operations on the real system unit record card readers.
	DMKRSTIN	Processes an interrupt from a real card reader, executable.
DMKSAD		Stand-alone, executable.
	DMKSADM	Produces a stand-alone dump of real storage on a tape or printer, executable.
	DMKSADWT	The utility that writes the stand-alone dump dump program on the IPL volume, executable.
DMKSAV		Pageable, executable. Saves and restores a page image count of the CP nucleus on the system residence disk.
	DMKSAVNC	Entered via an LDT card from DMKLDR. Writes a page image copy of the CP nucleus, executable.
	DMKSAVRS	Entered via a BALR from DMKCKP. Restores a page image copy of the CP nucleus, executable.
DMKSBL		Pageable, executable.
	DMKSBLTR	Creates a line of small block letters for DMKSEP, executable.

Module Name	Entry Points	Attributes, Function
DMKSCH		Resident, executable. Maintains queues of runnable and eligible users, alters the dispatching status of users, and periodically recalculates the working set size and deadline priority of users. DMKSCH contains the routines that maintain the system TOD clock comparator request queue and the code that monitors users with abnormal execution.
	DMKSCHAE	Processes the interrupt occurring when the favored execution measurement interval expires, executable.
	DMKSCHAP	Assured execution percentage for current favored execution user, non-executable.
	DMKSCHAU	Pointer to VMBLOK of favored user, non-executable.
	DMKSCHCA	Smoothed APU utilization field, non-executable.
	DMKSCHCO	Smoothed core utilization field, non-executable.
	DMKSCHCP	Interruption from real CPU timer, executable.
	DMKSCHCU	Smoothed CPU utilization field, non-executable.
	DMKSCHDL	Alters a users dispatching status, executable.
	DMKSCHMD	Interruption for the midnight date change, executable.
	DMKSCHN1	Current number of users in Q1 (interactive), non-executable.
	DMKSCHN2	Current number of users in Q2 (non-interactive), non-executable.
	DMKSCHPG	Desirable paging overhead percentage (< =99), non-executable.
	DMKSCHQ1	Q1 time slice, non-executable.
	DMKSCHQ2	Q2 time slice, non-executable.
	DMKSCHRL	Anchor for list of dispatchable users, non-executable.
	DMKSCHRT	Resets a clock comparator interruption request, executable.
	DMKSCHSC	Scheduler contention ratio field, non-executable.
	DMKSCHST	Establishes a clock comparator interruption request, executable.
	DMKSCHS1	Smoothed Q1 value, non-executable.
	DMKSCHS2	Smoothed Q2 value, non-executable.
	DMKSCHTI	At end of performance interval, update system performance indicators, executable.
	DMKSCHTQ	Anchor for list of outstanding clock comparator interrupt requests, non-executable.
	DMKSCH80	Interruption for real timer at storage address 80, executable.

Module Name	Entry Points	Attributes, Function
DMKSCN		Resident, executable. Scans module.
	DMKSCNAU	Searches the chain of VMBLOKs for one whose whose userid matches the one pointed to by register one, executable.
	DMKSCNDC	Returns the addresses of the RDEVBLOK that is given the device code (the D of CCPP or PPPD), executable.
	DMKSCNFD	Finds the next field in an input message buffer, executable.
	DMKSCNMI	Referenced for modification of assist, executable.
	DMKSCNMU	Convert RDEVBLOK for given SYSOWN device code, executable.
	DMKSCNEP	Determines if there is an online path from either processor, executable.
	DMKSCNPH	Calculates a bit mask defining the indicated device path, executable.
	DMKSCNRA	Computes a full real device address (in cuu form) from the RDEVADD, RCUADD, and TCHADD entries in the real device, control unit, and channel blocks, executable.
	DMKSCNRD	Computes a real device address (in CW form), from the RDEVADD, RCUADD, and RCHADD entries in the real device, control unit, and channel blocks, executable.
	DMKSCNRN	Returns the name of the real device to the caller in register 1, executable.
	DMKSCNRU	Returns the addresses of the real channel, control unit, and device blocks for a given real device to to the caller, executable.
	DMKSCNVD	Computes a full virtual device address (in cuu form), plus the addresses of the virtual channel and control unit blocks from a specific virtual device block, executable.
	DMKSCNVN	Returns the name of the virtual device to the caller in R1, executable.
	DMKSCNVS	Searches all the real device blocks for a device whose volume serial number matches the one pointed to by R1, executable.
	DMKSCNVU	Returns the addresses of the virtual channel, control unit, and device blocks for a given real device to the caller, executable.
DMKSCO		Pageable, executable.
	DMKSCOLI	Searches the logged on virtual machines for any links to a specified minidisk. A link is any virtual device whose RDEVBLOK pointer and relocation factor match those specified, executable.
	DMKSCONP	Finds the RCHBLOK and RCUBLOK that represents the next logical path to the device, executable.
DMKSEG		Pageable, executable.
	DMKSEGPG	Initializes CP page, segment, core, and swap tables, executable.
	DMKSEGWR	Creates page image copies of all the pageable modules between DMKSAV and DMKCKP, executable.

Module Name	Entry Points	Attributes, Function
DMKSEP		Pageable, executable.
	DMKSEPSP	Prints and punches the respective output separators on real spooling devices, executable.
	DMKSEPTL	Prints the trailer page, executable.
DMKSEQ		Resident, non-executable.
	DMKSEQDA	Contains a data area, non-executable.
	DMKSEQLA	Contains printer separator logo, non-executable.
	DMKSEQSA	The area for the sequence number, non-executable.
DMKSEV		Pageable but locked, executable.
	DMKSEV70	Analyzes 2870 channel logout and sets appropriate bits in the ECSW field according to the results of analysis. It moves the channel logout to the check record, executable.
DMKSIX		Pageable but locked, executable.
	DMKSIX60	Analyzes 2860 channel logout and sets appropriate bits in the ECSW field according to the results of analysis. It moves the channel logout to the check record, executable.
DMKSLC		Resident, executable. Set location counter for V=R system.
DMKSNC		Pageable, executable.
	DMKSNCP	Save a page-form version of a 3704/3705 network control program. The name of the network control program and the DASD location at which it is to be saved is defined in the CP module DMKSYS, executable.
DMKSND		Pageable, executable.
	DMKSNDNH	Processes the SEND command (which is used to send commands) and replies to disconnected virtual machines, executable.
DMKSNT		Pageable, non-executable. This module is assembled by the installation system programmer.
	DMKSNTBL	Label marking the start of the NAMESYS macro entries. The NAMESYS macro is used to describe systems to be saved via the SAVESYS command. Shared segments may be specified. These segments consist of all reenterable code and no altering of this storage is allowed, non-executable.
	DMKSNTLA	Label marking the start of the NAMELANG entries. NAMELANG entries are used to reserve DASD space for CP message repositories, non-executable.
	DMKSNTRN	Label marking the start of the NAMENCP macro entries, non-executable.
	DMKSNTQN	Label marking the start of the NAME3800 entries, non-executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKSPK		Pageable, executable.
	DMKSPKDL	Deletes used files from the system and de-allocates the DASD page space, executable.
	DMKSPKDR	This routine when, started by the CPEXBLOK, deletes all SFBLOKs in the DMKRSPDL chain and exits to dispatcher, executable.
DMKSPL		Pageable, executable. Spool file manager.
	DMKSPLCR	Closes and queues a real reader spool file for virtual input, executable.
	DMKSPLCV	Closes and queues a virtual printer or punch spool file for processing, executable.
	DMKSPLOR	Initializes control blocks and buffers for real input reader files, executable.
	DMKSPLOV	Initializes control blocks and buffers for virtual printer and punch output spool files, executable.
	DMKSPLSP	Locates the users VMBLOK, locates a virtual reader and sets device end pending and exits via CLOSEXIT, executable.
DMKSPM		Pageable, executable.
	DMKSPMEP	Processes the CP SPMODE command. Turns the single processor mode environment on and off, executable.
DMKSPS		Pageable, executable.
	DMKSPSIO	Performs the processing requested by the SPTAPE command and handles all returns from the I/O interrupt handler due to the command, executable.
DMKSPT		Pageable, executable.
	DMKSPTEP	Validates the format of the SPTAPE command and initiates the processing to write, read, or scan a tape for specified printer and punch spool files, executable.
DMKSRM		Pageable, executable.
	DMKSRMQS	Processes the QUERY SRM command, executable.
	DMKSRMSS	Processes the SET SRM command, executable.
DMKSSP		Standalone, executable. This module is found in the starter system only. It builds RCHBLOKs, RCUBLOKs, and RDEVBLOKs necessary to configure a minimum CP system. From the starter system, a real CP system figured based on the REALIO deck of the installation.
	DMKSSP01	Entered as a result of an IPL operation. Constructs the I/O blocks and system modules for a minimum system configuration, executable.

Module Name	Entry Points	Attributes, Function
DMKSSS		Resident, executable. Services routines for all other modules that require access to the MSS.
	DMKSSSAS	Attaches a 3330V to the system with a VOLID, executable.
	DMKSSSCA	The communicator device address, non-executable.
	DMKSSSCV	The VMBLOK address of the communicator virtual machine, non-executable.
	DMKSSSDE	Demounts an MSS volume from a 3330V, executable.
	DMKSSSEN	Returns to the appropriate requesting routine after an MSS volume mount is complete, executable.
	DMKSSSHR	The CCPD of the SDG table containing shared VUAs, non-executable.
	DMKSSSLN	Allocates a 3330V device and mounts the required 3330V system volume, executable.
	DMKSSSL1	Processes a DEDICATE statement with the 3330V parameter, executable.
	DMKSSSL2	Processes a DEDICATE statement with raddr and valid specified, and the raddr is a 3330V, executable.
	DMKSSSL3	Processes a DEDICATE statement with a volid but no raddr, executable.
	DMKSSSMQ	Serves as the anchor for the MSSCOM control blocks that are queued for MSS mounts, and pack change interruptions, non-executable.
	DMKSSSNS	The CCPD of the SDG table containing nonshared VUAs, non-executable.
	DMKSSSNV	The anchor of the SDG in which a VUA was last selected for a mount request, non-executable.
	DMKSSSRL	Issue a relinquish request to de-stage any changed cylinders of the volume mounted on the specified VUA, executable.
	DMKSSSSQ	End of queue pointer, non-executable.
	DMKSSSVA	Attaches a 3330V to the system or to a virtual machine, executable.
	DMKSSSVM	The userid of the communicator virtual machine, non-executable.
DMKSST		Pageable, executable. This is a service routine for modules that require access to the MSS.
	DMKSSTBL	Builds SDG tables of VUAs in CP configuration, executable.
	DMKSSTFV	Finds an available VUA on which to mount a virtual volume, executable.

Module Name	Entry Points	Attributes, Function
DMKSSU		Resident, executable. Handles service requests for processing whenever an attention interrupt or a cylinder fault is detected on a 3330V and whenever a reset is required for a virtual device defined on a 3330V
	DMKSSUCF	Resets a virtual device defined on a 3330V, including purging any I/O waiting for an MSS volume mount, executable.
	DMKSSUI1	Reschedules an I/O operation that had previously caused a cylinder fault, executable.
	DMKSSUI2	Queues an I/O request that has just caused a cylinder fault. Sets the missing attention handler timer interrupt value, executable.
	DMKSSULO	Checks for unfinished MSS processing before completing logoff, executable.
DMKSSV		Pageable, executable. This is a service routine for the MSS communicator virtual machine.
	DMKSSVHV	Process DIAGNOSE Code X'78', executable.
	DMKSSVUS	Quiesces all MSS mount and demount activity, executable.
DMKSTA		Pageable, executable.
	DMKSTABD	Used by DMKCPI to reference bad addresses, executable.
	DMKSTANT	Clears main storage, initializes the CORTABLE, and allocates main storage, executable.
DMKSTD		Resident, non-executable
	DMKSTDAT	Starting address of STDATA table, non-executable.
DMKSTK		Resident, executable. Stacks I/O blocks.
	DMKSTKCP	Stacks a CPEXBLOK, executable.
	DMKSTKDE	Stacks a deferred execution block, executable.
	DMKSTKIO	Stacks an IOBLOK, executable.
	DMKSTKLF	Stacks a CPEXBLOK LIFO, executable (used by EXTEND and machine check).
	DMKSTKMP	Stacks CPEXBLOK for current processor only, executable.
	DMKSTKOP	Stacks CPEXBLOK for the other processor only, executable.
	DMKSTKSW	Stacks a priority CPEVBLOK for other processor, executable.
DMKSTP		Pageable, executable.
	DMKSTPX	Updates the system performance indicators and scheduling control fields, executable.
DMKSTR		Pageable, executable.
	DMKSTRAN	Invoked from DMKPRT if there is a segment exception, executable.
	DMKSTRPM	Pseudo memory address translator, executable.
	DMKSTRSC	Counters for monitor, non-executable.
	DMKSTRSM	Invoked if the dispatcher has a CPEXBLOK for swap table migration, executable.

Module Name	Entry Points	Attributes, Function
DMKSVC		Resident, executable. Handles SVC interrupts generated while in the Supervisor State.
	DMKSVCHI	Contains DMKFRELI, non-executable.
	DMKSVCIN	SVC Interrupt Handler for Supervisor State, executable.
	DMKSVCLO	DMKFRELO minus length of one SAVEAREA, non-executable.
	DMKSVCNO	Number of SAVEAREAs to be obtained (pre processor), non-executable.
	DMKSVCNS	SVC parmlist used by CP assist - contains the address of the next available SAVEAREA, non-executable.
	DMKSVCN2	SVC parameter list for non-IPL procedure, non-executable.
DMKSVD		Resident, executable. Called via a goto from DMKSVC.
	DMKSVDIN	SVC Interrupt Handler for problem state, executable.
DMKSYM		Pageable, non-executable.
	DMKSYMTB	Provides a symbol table of all CSECTS and entry points, non-executable.
DMKSYS		Resident, non-executable. Exists as a CSECT that defines the system residence volume, paging space, operator ID, dump ID, storage size, and time zone.
DMKTAP		Pageable, executable. Examines the error condition resulting from a unit check while executing a CP generated tape channel program. Positioning of the tape is required on read/write commands and the channel program is re-executed. If the error condition is uncorrectable, a call is issued to the message writer (DMKMSW) to notify the operator. Upon regaining control from DMKMSW, the original channel program may be reexecuted or terminated.
	DMKTAPER	Retries the failing tape channel program, after a tape positioning command has been executed, executable.
	DMKTAPRL	Performs tape release to determine two- or four-channel switch capability, executable.
DMKTAQ		Pageable, executable. Continues tape error recovery started by DMKTAP.
	DMKTAQRE	Entered following a tape reposition operation, executable.
	DMKTAQRP	Repositions tape following a read type error, executable.
	DMKTAQSE	Continues checking for the cause of the original device error, executable.

Module Name	Entry Points	Attributes, Function
DMKTBL		Resident, non-executable. Contains the terminal translate tables.
	DMKTBLCO	EBCDIC to correspondence terminal code, non-executable.
	DMKTBLCI	Correspondence terminal code to EBCDIC, non-executable.
	DMKTBLGR	EBCDIC to 3270 APL code local and remote, non-executable.
	DMKTBLGT	EBCDIC to 3270 text code local and remote, non-executable.
	DMKTBLPI	PTTC/EBCD terminal code to EBCDIC, non-executable.
	DMKTBLPO	EBCDIC to PTTC/EBCD terminal code, non-executable.
	DMKTBLRG	EBCDIC to 3270 code local and remote, non-executable.
	DMKTBLSF	Remove start field orders and attributes, non-executable.
	DMKTBLUP	Lower-case EBCDIC to upper-case EBCDIC, non-executable.
DMKTBM		Pageable, non-executable. Contains terminal translate tables for APL/ASCII for non-TTY terminals.
	DMKTBMAI	APL 3278 compound read translation, non-executable.
	DMKTBMAO	APL 3278 compound write translation, non-executable.
	DMKTBMMI	APL Correspondence terminal code -> EBCDIC, non-executable.
	DMKTBMMO	EBCDIC -> APL correspondence terminal code, non-executable.
	DMKTBMNI	APL PTTC/EBCD terminal code -> EBCDIC, non-executable.
	DMKTBMNO	EBCDIC -> APL PTTC/EBCD terminal code, non-executable.
	DMKTBMTI	Text 3270 compound read translation, non-executable.
	DMKTBMTO	Text 3270 compound write translation, non-executable.
	DMKTBMXI	Text 3278 compound read translation, non-executable.
	DMKTBMXO	Text 3278 compound write translation, non-executable.
	DMKTBMZI	APL 3270 compound read translation, non-executable.
	DMKTBMZO	APL 3270 compound write translation, non-executable.
DMKTBN		Pageable, non-executable. Contains terminal translate tables for APL/ASCII for TTY terminals.
	DMKTBNAE	ASCII to EBCDIC translation, non-executable.
	DMKTBNAI	ASCII APL to EBCDIC APL translation, non-executable.
	DMKTBNAO	EBCDIC APL to ASCII APL translation, non-executable.
	DMKTBNEA	EBCDIC to ASCII translation, non-executable.
DMKTCS		Pageable, executable.
	DMKTCSCO	Sets up the forms overlay sequence control, executable.
	DMKTCSET	Sets up the 3800 printer prior to printing the file, executable.
	DMKTCSML	Loads members from image library, executable.
	DMKTCSSP	Sets up the 3800 printer prior to printing the separator, executable.
	DMKTCSTR	Prints trailer page on a 3800 printer, executable.
DMKTCT		Pageable, executable.
	DMKTCTET	Loads character arrangement table, WCGMs, LCSs, and graphic character modifications into a 3800 printer, executable.

Module Name	Entry Points	Attributes, Function
DMKTDK		Pageable, executable.
	DMKTDKGT	Allocates cylinders of temporary disk space from cp-owned volumes, executable.
	DMKTDKRL	Releases temporary disk space to the pool of free space, executable.
DMKTHI		Pageable, executable. Displays data about use of and contention for major system resources.
	DMKTHIFA	Processes INDICATE FAVORED command, executable.
	DMKTHIIO	Processes INDICATE I/O command, executable.
	DMKTHILO	Processes INDICATE LOAD command, executable.
	DMKTHIPA	Processes INDICATE PAGING command, executable.
	DMKTHIQQ	Processes INDICATE QUEUES command, executable.
	DMKTHIUG	Processes INDICATE USER command for general users, executable.
	DMKTHIUS	Processes INDICATE USER command for primary system operators, system resource operators, and system analysts, executable.
DMKTMR		Resident, executable. Simulates the CPU timer and time-of-day clock comparator instructions for virtual machines operating in EC mode.
	DMKTMRCC	Entered after expanded virtual machine assist processing of a virtual SCKC instruction, executable.
	DMKTMRCK	Simulates virtual clock comparator interruptions, executable.
	DMKTMRFR	Entry to clean up storage subpools, executable.
	DMKTMRPT	Calculates users total virtual problem time, executable.
	DMKTMRSN	Entry to update total VMTTIME used by processors, executable.
	DMKTMRSP	Entered after expanded virtual machine assist processing of a virtual SPT instruction, executable.
	DMKTMRTN	Simulates timer instruction, executable.
	DMKTMRVT	Simulates virtual CPU timer interruptions, executable.
DMKTOD		Pageable, executable.
	DMKTODIN	Initializes the time of day clock, executable.
	DMKTODTB	Buffer used by various initialization routines, executable.
DMKTPE		Pageable, executable.
	DMKTPERP	Performs 3480 error recovery processing, executable.
DMKTRA		Pageable, executable. Processes the TRACE command line. Provides a virtual machine with facility to track SVC instructions, program interrupts, external interrupts, successful searches, or all instructions with output on the printer or terminal
	DMKTRACE	TRACE command processor, executable.

Module Name	Entry Points	Attributes, Function
DMKTRC		Pageable, executable. Processes the TRACE command functions.
	DMKTRCEX	Traces external interruptions, executable.
	DMKTRCIO	Traces I/O interruptions, executable.
	DMKTRCIT	Sets the needed SVC B2 for instruction tracing, executable.
	DMKTRCND	Ends tracing, executable.
	DMKTRCPB	Puts back user instructions altered by tracing, executable.
	DMKTRCPG	Traces program interruptions, executable.
	DMKTRCPV	Traces privileged instruction interruptions, executable.
	DMKTRCSV	Processes an SVC, Branch, or full instruction TRACE, executable.
	DMKTRCSW	Traces virtual and real CSWs, executable.
DMKTRD		Pageable, executable.
	DMKTRDSI	Traces I/O operations (SIO, TIO, HIO, TCH), executable.
	DMKTRDWT	Serialization entry for I/O and CCW tracing, executable.
DMKTRK		Resident, executable.
	DMKTRKFP	Examines Command Rejects for virtual SIO, executable.
	DMKTRKIN	Handles interrupts caused by alternate tracks, executable.
	DMKTRKVA	Verifies alternate track address for DMKCCW, executable.
DMKTRM		Pageable, executable.
	DMKTRMID	Identifies a 2741 terminal as either a 2741P (PTTC/EBCD) or 2741C (correspondence) from the user command. It sets ADEVTYPE the RDEVBLOK to TYP2741P or TYP2741C and sets flag RDEVIDNT on if the terminal was successfully identified, executable.
DMKTRP		Pageable, executable.
	DMKTRPRE	Stops CPTRAP processing due to LOGOFF, executable.
	DMKTRPST	Processes the CPTRAP command line, executable.
DMKTRT		Pageable, executable.
	DMKTRTCM	Common processing routine to enter data into a CPTRAP file, executable.
DMKTRU		Pageable, executable.
	DMKTRUAC	Activates the CPTRAP facility, executable.
	DMKTRUST	Stops an active CPTRAP facility, executable.
DMKTRX		Pageable, executable.
	DMKTRXCP	Interface to the CPTRAP facility to record data generated by CP code, executable.
	DMKTRXEX	Exit routine for the CPTRAP Interface entry points, executable.
	DMKTRXLK	CPTRAP processing lock, non-executable.
	DMKTRXTT	Interface to the CPTRAP facility to record internal CP trace table entries, executable.
	DMKTRXVT	Interface to the CPTRAP facility to record data generated by virtual machine code, executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKTTX		Pageable, executable.
	DMKTTXIN	Handles 3101 program function keys and password overlay. Causes the input data to be stored in retrieve buffer, executable.
	DMKTTXTR	Translates input (READ) and output (WRITE) CONTASK data from ASCII to EBCDIC and from EBCDIC to ASCII 7-bit line code. Scans for shift in and shift out characters when APL is on, executable.
DMKTTY		Pageable, executable.
	DMKTTYIN	Entry for input CONTASKs, translates CONTASK data from ASCII 7-bit line code to EBCDIC, overlays passwords with blanks, handles 3101 PF keys, causes input data to be stored in retrieve buffer, construct more CONTASK when scroll value reached, executable.
	DMKTTYOP	Handles prompting display for TTY or VM option. Determines 3101 output displayed location and compresses blanks in 3101 data streams, executable.
DMKTTZ		Resident, non-executable.
	DMKTTZLF	Contains CCWs and data pointed to by certain CCWs for TTY terminals, non-executable.
DMKUCB		Pageable, non-executable.
	DMKUCBLD	Contains the UCB buffer load images used by the LOAD command to load the universal character set buffer in the 3811 control unit, non-executable.
DMKUCC		Pageable, non-executable.
	DMKUCCLD	Contains the UCB buffer load images used by the LOAD command to load the universal character set buffer in the 3203 printer control unit, non-executable.
DMKUCS	DMKUCSLD	Pageable, non-executable. Contains the UCS buffer load images that the LOAD command uses to load the universal character set buffer in the 2821 control unit, non-executable.

Module Name	Entry Points	Attributes, Function
DMKUDR		Pageable, executable.
	DMKUDRBV	Allows the DMKDIRCT or DMKCPINT programs to build a list of virtual page buffers, one for each UDIRBLOK page on disk, executable.
	DMKUDRDS	Allows the DMKDIRCT program to swap the active user directory to the newly created user directory, executable.
	DMKUDRFD	Puts specified UDEVBLOK into the caller's buffer, executable.
	DMKUDRFU	Finds a given user ID in the user directory and moves the user's directory entry into the caller's buffer, executable.
	DMKUDRIA	Finds IUCV directory authorization, executable.
	DMKUDRMD	Reads the account block, executable.
	DMKUDROV	Called during system initialization by DMKCPI to read the Class Override records saved on the Directory cylinder area, executable.
	DMKUDRRD	Reads the next user directory into the caller's buffer, executable.
	DMKUDRRV	Releases a virtual page used by the directory program as a buffer, executable.
	DMKUDRXI	Reads the UIPLBLOK into the caller's buffer, executable.
DMKUDU		Pageable, executable.
	DMKUDUMN	Updates in-place the CP directory on the object DASD page and updates in-place the virtual system page (if used) on the paging device. Entered from from DMKHVD when a class B virtual machine issues a DIAGNOSE Code X'84' instruction, executable.
DMKUNT		Resident, executable. Untranslates CCWs and CSWs.
	DMKUNTFB	Relocates sense-byte information for the FB-512 devices. The virtual physical address from sense bytes 3-6 is computed from the real physical address given by the hardware. For formats 0 4, sense bytes 18-21 and for format 1 message A, bytes 8-10 are also computed from the real physical address given by the hardware, executable.
	DMKUNTFR	Releases pages and free storage used for the CCW chain. Also processes the CP assist instruction, UNTFR (E605), non-executable.
	DMKUNTF1	CP assist pointers, non-executable.
	DMKUNTIS	Finds the RCWTASKS that have been dispatched to handle OS ISAM self-modifying sequences and put them back the way DMKCCW had them to allow DMKUNTRN and DMKUNTFR to operate correctly, executable.
	DMKUNTRN	Translates a real CSW into a virtual CSW. Also processes the CP assist instruction, UNTRN (E610), non-executable.
	DMKUNTRS	Relocates sense byte information. For a 3330, 3340, 3350, 3375, 3380, or 2305, computes virtual cylinder number in byte 5 and 6 of the sense data by unrelocating the real cylinder number given by the hardware. For a 2311 simulated on a 2314 or 2319, computes the appropriate status for byte 3 of the sense data from the real sense data given by the hardware, executable.

Module Name	Entry Points	Attributes, Function
DMKURS		Pageable, executable.
	DMKURSTA	Displays status messages for real unit record devices, executable.
DMKUSO		Pageable, executable. Processes user termination.
	DMKUSODS	Processes the DISCONN (disconnect) command, executable.
	DMKUSOFF	Begins processing to LOGOFF a user, executable.
	DMKUSOFL	Processes the FORCE command, executable.
	DMKUSOLG	Processes the LOGOFF command, executable.
DMKUSP		Pageable, executable.
	DMKUSPFM	Returns subpools from the free storage chain and removes spool file blocks from the dynamic paging area, executable.
DMKUSQ		Pageable, executable.
	DMKUSQFF	Continues processing to LOGOFF a user, executable.
DMKVAT		Resident, executable. Storage management for EC mode virtual machine.
	DMKVATAB	Allocates, initializes and maintains shadow segment, and page tables for virtual machines that can relocate, executable.
	DMKVATAT	External reference to tables, executable.
	DMKVATBC	Returns active shadow tables to free storage, executable.
	DMKVATEX	Services page or segment exceptions for virtual EC machines, executable.
	DMKVATMD	Allocates and initializes shadow tables, executable.
	DMKVATOF	Executable.
	DMKVATPF	Handles pseudo page fault interruption from a VS1 virtual machine, executable.
	DMKVATPX	Processes paging exceptions for a virtual machine that performs paging, executable.
	DMKVATSI	Selectively invalidate shadow page table entries, executable.
	DMKVATSX	Processes segment exception for a virtual machine that performs paging, executable.
	DMKVATZP	Processes the CP assist instruction, ZAPPAGE (E60B). executable.
	DMKVATZS	Processes the CP assist instruction, (E60A), executable.
DMKVAU		Resident, executable.
	DMKVAUAT	External reference to tables, executable.
	DMKVAULA	Virtual; virtual-to-virtual address translation, executable.
	DMKVAURN	Virtual (shadow); virtual-to-real address translation, executable.
	DMKVAUZP	Non-executable.

Module Name	Entry Points	Attributes, Function
DMKVBM		Pageable, executable.
	DMKVBMIN	Initializes the virtual buffer space. This involves creating an address space, putting the new VMBLOK associated with this address space on the buffer manager chain, and building the segment, page, and allocation tables needed to manage the address space, executable.
	DMKVBMSC	Scans the buffer manager chain for a requested VMBLOK, executable.
	DMKVBMVG	Gets a page of virtual memory from the specified address space; marks the page and cylinder where this buffer was allocated, executable.
	DMKVBMVM	Builds a VMBLOK for system paging, executable.
	DMKVBMVR	Returns a page of virtual memory to the specified address space; de-allocates the page and cylinder where the buffer resided, executable.
DMKVCA		Pageable, executable. Simulates I/O for a virtual channel-to-channel device (CTCA or 3088).
	DMKVCAST	Simulates the channel and device operations of the channel-to-channel device connected between two virtual machines under VM/SP, executable.
DMKVCB		Pageable, executable. Simulates I/O for a virtual channel-to-channel adapter.
	DMKVCBRD	Selectively resets a device for a virtual channel-to-channel device without decoupling the CTC device from the Y-side adapter, executable.
	DMKVCBRS	Does a final reset for a virtual channel-to-channel device and disconnects the device from its coupled twin on the Y-side virtual machine, executable.
	DMKVCBSH	Simulates the execution of a HALT I/O or HALT DEVICE instruction for a virtual machine channel-to-channel device, executable.
	DMKVCBTS	Simulates the TEST I/O instruction for a virtual channel-to-channel device that has no interruptions pending, executable.
DMKVCH		Pageable, executable.
	DMKVCHDC	Processes the ATTACH and DETACH real devices and channels command, executable.

Module Name	Entry Points	Attributes, Function
DMKVCN		Resident, executable.
	DMKVCNEX	Simulates all SIOs to a virtual console, executable. If a larger than 4K buffer is required, this module: • Gets the number of pages necessary for the noncontiguous buffer from the page manager, and locks the pages in storage. • Gets and builds an indirect address list from free storage, and uses the indirect address list to address the noncontiguous buffer. • Returns the pages of the buffer via the address list space to free storage after the operation is complete.
DMKVCP		Resident, executable.
	DMKVCPIL	Processes the function specified in the control area of an IUCV SEND request, executable.
DMKVCQ		Resident, executable.
	DMKVCQAT	TTY ATTENTION (BREAK Key) processing, executable.
	DMKVCQEX	Writes exclamation points for TTY break key processing, executable.
	DMKVCQRE	Processes the reply to write request, executable.
	DMKVCQSR	Processes the asynchronous passback from a send, executable.
DMKVCR		Resident, executable.
	DMKVCRNR	Sends any pending batched writes to the VSM, executable.
	DMKVCRMT	Processes the reply to the read. At completion of full screen read, when an indirect address list is being used, data is moved from WEBLOK to storage using the indirect address list, executable.
	DMKVCRRD	Requests the VTAM Communications Network Application (VSCS) to perform a read operation, executable.
DMKVCS		Resident, executable.
	DMKVCSMO	Return from IUCV, executable.
	DMKVCSWT	Requests VSCS to perform a write operation, executable.
DMKVCT		Resident, executable.
	DMKVCTCH	Processes color attribute changes for SNA for SNA logical units, executable.
	DMKVCTCU	Processes the accounting data, executable.
	DMKVCTDA	Disables VSM access to CP, executable.
	DMKVCTEN	Enables VSM access to CP, executable.
	DMKVCTER	Processes 'TERM' command change for LUs, executable.
	DMKVCTLO	Releases SNA CCS control blocks and process accounting data, executable.
	DMKVCTTM	Sends a request to VSCS to redisplay the input line, executable.
DMKVCU		Resident, executable. VM/VTAM second level interrupt handler.
	DMKVCUIL	Processes asynchronous service requests, executable.

Module Name	Entry Points	Attributes, Function
DMKVCV		Resident, executable.
	DMKVCVCE	Writes trace entry for the CCS transaction, executable.
	DMKVCVEB	Builds WEBLOK. When obtaining WEBLOK for a full screen write, an indirect address list is used to move data to WEBLOK, executable.
	DMKVCVER	Sets redisplay timer, executable.
	DMKVCVIN	Releases 'read' control blocks, executable.
	DMKVCVIX	Build IXBLOK and do IUCV SEND to VSCS, executable.
	DMKVCVKS	Locates control blocks, executable.
	DMKVCVLD	Builds and chain a WEBLOK, executable.
	DMKVCVLY	Issues an IUCV reply for a two-way send, executable.
	DMKVCVND	Issues IUCV send for unsent writes/reads, executable.
	DMKVCVUT	Releases 'write' control blocks, executable.
DMKVCW		Resident, executable.
	DMKVCWCN	Initiates the communication path for the VTAM Service Machine and VTAM Logical Units and sets up the initial environment, executable.
	DMKVCWQS	Suspends activity for the user's virtual machine, executable.
	DMKVCWRM	Terminal characteristics, executable.
	DMKVCWSV	Breaks the IUCV communication path, executable.
DMKVCX		Resident, executable.
	DMKVCXD2	Process abend due to invalid control block chain, executable.
	DMKVCXFU	Error handler for IUCV transactions, executable.
	DMKVCXGF	Begin logoff of user, executable.
	DMKVCXGO	Build WEBLOK and move in the system logo, executable.
	DMKVCXIO	Process unrecoverable I/O error, executable.
	DMKVCXOR	Send the color attribute for the LU to the VSCS, executable.
	DMKVCXOX	Do one-way IUCV SEND of logo to VSCS, executable.
	DMKVCXSA	Process VSCS logic error, executable.
DMKVDA		Pageable, executable.
	DMKVDAAA	Processes the ATTACH command, executable.
	DMKVDAAC	Processes the ATTACH CHANNEL command, executable.
DMKVDB		Pageable, executable.
	DMKVDBMD	Processes ATTACH messages for DMKVDA when multiple device addresses are processed, executable.
DMKVDC		Pageable, executable.
	DMKVDCPS	Acquires virtual blocks for devices that are likely to be attached by the ATTACH command, executable.
	DMKVDCSC	Scans the ATTACH and DETACH command lines and checks syntax, executable.
DMKVDD		Pageable, executable.
	DMKVDDDC	Processes the DETACH CHANNEL command for general users, executable.
	DMKVDDDG	Processes the DETACH command for general users. executable.
	DMKVDDDR	Processes the DETACH command for system resource operators, executable.
	DMKVDDDS	Processes the DETACH CHANNEL command for system resource operators, executable.

Module Name	Entry Points	Attributes, Function
DMKVDE		Pageable, executable.
	DMKVDEDC	Verifies the existence of a device specified on an ATTACH command, executable.
	DMKVDERR	Subroutine to issue attach error messages, executable.
DMKVDF		Pageable, executable.
	DMKVDFRE	Processes the error messages detected in the DETACH command, executable.
DMKVDG		Pageable, executable.
	DMKVDGAL	Calculates the size of the preferred fixed or moveable head areas on a system-owned DASD that is varied online and attached to the system, executable.
DMKVDR		Pageable, executable.
	DMKVDREL	Releases a virtual or real device from a virtual user, where the virtual device is to be detached, executable.
	DMKVDRES	Releases a virtual or real device from a virtual user, where the virtual device is not to be detached, executable.
DMKVDS		Pageable, executable.
	DMKVDSAT	Attaches a virtual device to a user, executable.
	DMKVDSDF	Defines a new virtual device for user, executable.
	DMKVDSLK	Links a virtual DASD device to a user, executable.
	DMKVDSND	Shows the end of DMKVDS on load map, executable.
DMKVER		Pageable, executable. Processes error records from virtual machine via SVC 76.
	DMKVERD	Processes SVC 76 from DOS or CMS/DOS, executable.
	DMKVERO	Processes SVC 76 from OS, VS/1, VS/2, VM/370, or VM/SP, executable.
DMKVIO		Resident, executable. Records and translates the interrupts and status associated with virtual I/O operations.
	DMKVIOCI	Count of virtual CLRIOs, non-executable.
	DMKVIOCL	Clears the VDEVIO queue, executable.
	DMKVIOCT	Count of total I/O instructions, non-executable.
	DMKVIOCW	Count of calls to DMKCCWTR, non-executable.
	DMKVIOC1	Reflects condition code 1 CSW status, executable.
	DMKVIOHD	Count of virtual HDVs, non-executable.
	DMKVIOHI	Count of virtual HIOs, non-executable.
	DMKVIOIN	Translate a virtual I/O interruption, executable.
	DMKVIOMK	Address of a table of interruption masks, indexable by device address, executable. Masks to set interrupt pending in VMBLOK.
	DMKVIOSF	Count of virtual SIOFs, non-executable.
	DMKVIOSI	Count of virtual SIOs, non-executable.
	DMKVIOTC	Count of virtual TIOs, non-executable.
	DMKVIOTI	Masks to reset interrupt pending in VMBLOK, non-executable.
	DMKVIOXK	Masks to reset interrupt pending in VMBLOK, non-executable.

Module Name	Entry Points	Attributes, Function
DMKVMA		Resident, executable.
	DMKVMACT	Counts the number of times DMKVMASH was entered. non-executable.
	DMKVMASH	Checks all protected shared pages associated with shared named systems and determines if they have been changed. If they were changed, the page is returned to CP free storage and the condition code is made nonzero, executable.
	DMKVMASW	Switches the user's segment table entries from one protected shared page table to the other, executable.
	DMKVMAS1	Anchor for shared systems, non-executable.
	DMKVMAS2	Anchor for named systems, non-executable.
DMKVMC		Pageable, executable.
	DMKVMCEX	Called by DMKDSP to reflect the VMCF external interrupt message header and optional SENDX data to a virtual machine. Copies the message header from the VMCBLOK to the user's external interrupt buffer. If interrupt is for a SENDX request, move SENDX data to the optional area in the external interrupt buffer, executable.
	DMKVMCFC	Main entry for all VMCF subfunctions. Called by DMKHVC when a DIAGNOSE X'0068'instruction is executed. Builds a VMCBLOK with information from user-supplied parameter list, validates the subfunction code, and passes control to appropriate VMCF subroutine, executable.
	DMKVMCUA	Branched to from the DMKVMCFC entry point or called by DMKCFP during a system reset. Releases the master VMCBLOK and any final response VMCBLOKs (VMCRESP bit). Returns other VMCBLOKs to the original SOURCE users with the notification that this user is not available, executable.
DMKVMD		Pageable, executable.
	DMKVMDEP	Dumps guest virtual machine to spool blocks in binary form. The output is read by DIAGNOSE X'14'. VM/IPCS or a user-written routine may be used to process the guest virtual machine dump, executable.
	DMKVMDIA	Entry point for DIAGNOSE X'94', executable.
DMKVME		Pageable, executable.
	DMKVMEDP	Provides the dumping service for the VMDUMP command and DIAGNOSE Code X'94', executable.
DMKVMG		Pageable, executable.
	DMKVMGCN	Handles pending CONNECTs from a virtual machine to the Signal system service, executable.
	DMKVMGIL	Handles pending messages representing signals for the Signal system service, executable.
	DMKVMGSV	Handles pending SEVERs from a virtual machine to the Signal system service, executable.

Module Name	Entry Points	Attributes, Function
DMKVMI		Pageable, executable.
		Loaded into the user's virtual storage when invoked. Performs an IPL of a virtual machine.
	DMKVMIPL	Simulates a user's IPL sequence, executable.
DMKVSC		Resident, executable.
	DMKVSCAN	Scans a V=R channel program for exceptional conditions, such as sense commands, NOOPs, I/O to and from page 0, etc., without actually translating the program, executable.
	DMKVSCVR	Scans for conditions indicating NOTRANS is a valid operation, executable.
DMKVSI		Resident.
		Simulates the operation of privileged I/O instructions issued by virtual machines, executable.
	DMKVSICH	Re-entry from DMKVSJ to process CLCH as a TCH. executable.
	DMKVSICI	Count of virtual CLRIOs, non-executable.
	DMKVSICT	Count of total I/O instructions, non-executable.
	DMKVSICW	Count of calls to DMKCCWTR, non-executable.
	DMKVSIEX	Simulates a SIO, TIO, HIO, TCH, or CLCH, executable. (HIO and CLCH are processed by DMKVSJ.)
	DMKVSIFT	Re-entry from DMKVSJ to release the IOBLOK, executable.
	DMKVSISI	Count of virtual SIOs, non-executable.
	DMKVSISF	Count of virtual SIOFs, non-executable.
	DMKVSITI	Count of virtual TIOs, non-executable.
	DMKVSITC	Count of virtual TCHs, non-executable.
	DMKVSIVS	Entry for microcode assist only, executable.
DMKVSJ		Resident, executable.
	DMKVSJEX	Simulates the operation of the privileged I/O instructions HIO and CLCH issued by virtual machines, executable.
	DMKVSJHI	Count of virtual HIOs, non-executable.
	DMKVSJHD	Count of virtual HDVs, non-executable.
	DMKVSJCC	Count of virtual CLCHs, non-executable.
DMKVSP		Resident, executable.
		Simulates all user SIOs to a virtual unit device (real reader, punch, print, or pseudo timer) that is spooled rather than dedicated.
	DMKVSPEX	Simulates SIO to a spooled unit record device, executable.
	DMKVSPPE	Printer end-of-file processing, executable.
	DMKVSPST	Stacks a CPEXBLOK, executable.
	DMKVSPUS	Unstacks a CPEXBLOK, executable.
	DMKVSPWA	Index work area for 2311, non-executable.
DMKVSQ		Resident, executable.
	DMKVSQPD	Locates next available slot in a printer/punch DASD buffer and move CCW and data into that slot, executable.
DMKVSR		Resident, executable.
	DMKVSRGC	Finds and validates the next non-TIC CCW in the user's channel program, executable.
	DMKVSRMD	Locates user's data area and moves data between the user's area and the in-storage work buffer, executable.

Module Name	Entry Points	Attributes, Function
DMKVST		Resident, executable. Handles control program requests to print on the user's virtual printer.
	DMKVSTCP	Writes a print line to the console, executable.
	DMKVSTOP	Opens output spool file, executable.
	DMKVSTPT	Puts a CP-generated line on the user's spooled printer, executable.
	DMKVSTVP	Writes a console hardcopy print line, executable.
DMKVSU		Pageable, executable.
	DMKVSUCO	Stops processing the file currently in spooled printer or punch and clears all pending status from the spooled printer or punch, executable.
	DMKVSUCR	Stops processing the file currently in the spooled card reader and clears all pending status from the spooled card reader, executable.
DMKVSV		Resident, executable.
	DMKVSVLD	Verifies the validity of 3800 printer load CCWs and sets up the in-storage work buffer, executable.
DMSVSW		Resident, executable.
	DMKVSWDC	Clears device blocks, executable.
	DMKVSWFC	Starts clearing old reader files, executable.
	DMKVSWOC	Searches the reader file chain, executable.
	DMKVSWOR	Opens reader file, executable.
	DMKVSWOT	Reader file half open - finish open, executable.
	DMKVSWTO	TIO to spooled reader, executable.
DMKVSX		Resident, executable. Simulates all user SIOs to a virtual card reader or pseudo timer that is spooled rather than dedicated. Simulates sense CCWs for spooled unit record devices and sense ID CCWs for spooled printers.
	DMKVSXCL	Simulates a close of a spooled unit record reader, executable.
	DMKVSXRD	Locates the next data record in a reader file, executable.
	DMKVSXSE	Simulates a sense to a spooled unit record, executable.
	DMKVSXSI	Simulates a sense ID to a spooled printer, executable.
	DMKVSXSR	Simulates a SIO to a spooled unit record reader, executable.
	DMKVSXTR	Simulates a SIO to timer, executable.
DMKWRM		Pageable, executable.
	DMKWRMFB	Allocates spool and dump space for FB-512 during warm start, executable.
	DMKWRMST	Warm start processing. Retrieves the system log messages, accounting cards, spool file blocks, and spooling allocation records from the warm start cylinder on the IPL pack, executable.
DMKWRN		Pageable, executable.
	DMKWRNPL	Performs a checkpoint on any alterations in the spool file setup to allow the recovery routine to get them if warm start fails, executable.
	DMKWRNWN	Initializes the checkpoint area cylinders and checkpoints any SHQBLOCKs or SFBLOCKs, executable.

Module Name	**Entry Points**	**Attributes, Function**
DMKXAB		Pageable, executable.
	DMKXABDG	Processes the DIAGNOSE X'B4' and X'B8' instructions, executable.
DMKXAD		Pageable, executable.
	DMKXADVS	Copies an XAB (External Attribute Buffer) associated with a virtual printer to a spool file for that device, executable.
DMKZTD		Pageable, executable.
	DMKZTDDF	Clears the first CKD cylinder or first FB-512 block of a TDSK mini-disk if SYSCLR=NO was specified on the SYSRES macro, executable.
	DMKZTDST	Cleans CKD and FB-512 T-Disk Space, executable.

Part 4: CP Diagnostic Aids

This part contains the following information:

- Entry Points for CP Commands
- Function Codes for DIAGNOSE Instructions.

Chapter 9. Entry Points for CP Commands

The following is a list of CP commands and the modules that gain control to perform their functions:

Command	Entry Point(s)	Command	Entry Point(s)
ACNT	DMKCPVAC	FREE	DMKCSQFR
ADSTOP	DMKCFDAD	HALT	DMKCPSH
ATTACH	**	HOLD	DMKCSQH
ATTN	DMKCFJRQ	INDICATE	**
AUTOLOG	DMKALGON	IPL	DMKCFGIP
BACKSPAC	DMKCSFBS	LINK	DMKLNKIN
BEGIN	DMKCFJBE	LOADBUF	DMKCSBLD
CHANGE	DMKCSUCS	LOADVFCB	DMKCSBVL
	DMKCSUCG	LOCATE	DMKCFDLF
CLOSE	DMKCSQCL	LOCK	DMKCPVLK
COMMANDS	DMKCQSQC	LOGOFF	DMKUSOLG
COUPLE	DMKDIBCP	LOGON	DMKLOGON
CP	DMKCFM	MESSAGE	DMKMSGMS
CPTRAP	DMKTRPST	MIGRATE	DMKPGMUS
DCP	DMKCDBDC	MONITOR	DMKMCCCL
DEFINE	DMKDEFDS	MSGNOH	DMKMSGNH
	DMKDEFDG	NETWORK	**
DETACH	**	NOTREADY	DMKCPBNR
DIAL	DMKDIAL	ORDER	DMKCSVOS
DISABLE	DMKCPVDS		DMKCSVOG
DISCONN	DMKUSODS	PER	DMKPEINT
DISPLAY	DMKCDBDI	PURGE	DMKCSVPS
DMCP	DMKCDMDM		DMKCSVPG
DRAIN	DMKCSODR	QUERY	**
DUMP	DMKCDMDU	QVM	DMKQVMEP
ECHO	DMKMSGEC	READY	DMKCPBRY
ENABLE	DMKCPVEN	REPEAT	DMKCSFRP
EXTERNAL	DMKCPBEX	REQUEST	DMKCFJRQ
FLUSH	DMKCSFFL	RESET	DMKCPBRS
FORCE	DMKUSOFL	REWIND	DMKCPBRW

Command	**Entry Point(s)**
SAVESYS	DMKCFHSV
SCREEN	DMKCFWEP
SEND	DMKSNDNH
SET	**
SHUTDOWN	DMKCPSSH
SLEEP	DMKCFJSL
SMSG	DMKMSGSM
SPACE	DMKCSFSP
SPMODE	DMKCSPSP
SPOOL	DMKSPMEP
SPTAPE	DMKSPTEP
START	DMKCSOST
STCP	DMKCDSCP
STORE	DMKCDSTO
SYSTEM	DMKCPBSR
TAG	DMKCSTAG
TERMINAL	DMKCFTRM
TRACE	DMKTRACE
TRANSFER	DMKCSVTS
	DMKCSVTG
UNLOCK	DMKCPVUL
VARY	DMKCPTNF
VMDUMP	DMKVMDEP
WARNING	DMKMSGWN
*	DMKCFM

** DMKCFC uses a subcommand table in DMKCMD to find the entry points for the commands ATTACH, DETACH, INDICATE, NETWORK, QUERY, and SET. See the following chart.

Command	**Subcommand**	**Function Type**	**Entry Label in DMKCMD**	**Final Entry Label**
ATTACH		R	DMKCMDAT	DMKVDAAA
ATTACH	CHANNEL	R		DMKVDAAC
DETACH		R	DMKCMDDE	DMKVDDDR
DETACH	CHANNEL	R		DMKVDDDS
DETACH		G	DMKCMDDG	DMKVDDDG
DETACH	CHANNEL	G		DMKVDDDC
INDICATE	FAVORED	A,O	DMKCMDIN	DMKTHIFA
	I/O	A		DMKTHIIO
	PAGING	A		DMKTHIPA
	QUEUES	A		DMKTHIQQ

Command	Subcommand	Function Type	Entry Label in DMKCMD	Final Entry Label
	USER	A		DMKTHIUS
	LOAD	A,G	DMKCMDIG	DMKTHILO
	USER	G		DMKTHIUG
NETWORK	SHUTDOWN	O	DMKCMDNO	DMKNESHD
	ATTACH	O,R	DMKCMDNR	DMKNEAAH
	DETACH	O,R		DMKNEADH
	DISABLE	O,R		DMKNETDB
	DISPLAY	O,R		DMKNESDS
	DUMP	O,R		DMKNLEMP
	ENABLE	O,R		DMKNETEN
	LOAD	O,R		DMKNLDR
	POLLDLAY	O,R		DMKNESPL
	QUERY	O,R		DMKNETQU
	VARY	O,R		DMKNETVA
QUERY	ALL	R	DMKCMDQR	DMKCQPQA
	DASD	R		DMKCQPQD
	DUMP	R		DMKCQPDP
	GRAF	R		DMKCQPQG
	LINES	R		DMKCQPQL
	MITIME	R		DMKCQSMI
	STATUS	R		DMKCQTST
	STORAGE	R		DMKCQPQS
	SYSTEM	R		DMKCQQQS
	TAPES	R		DMKCQPQT
	TDSK	R		DMKCQQQT
	UR	R		DMKCQPQU
	FILES	S	DMKCMDQS	DMKCQHFS
	HOLD	S		DMKCQRHD
	PRINTER, PRT	S		DMKCQHTS
	PUNCH, PCH	S		DMKCQHNS
	READER, RDR	S		DMKCQHRS
	AFFINITY	A,O	DMKCMDQA	DMKCQRAF
	CPASSIST	A,O		DMKCQYCA
	JOURNAL	A,O		DMKJRLQU
	PAGING	A,O		DMKCQRPG
	PRIORITY	A,O		DMKCQRPR
	QDROP	A,O		DMKCQRQD
	SASSIST	A,O		DMKCQYSA
	SRM	A		DMKSRMQS
	CPLANG	all	DMKCMDQG	DMKCQTCL

Command	Subcommand	Function Type	Entry Label in DMKCMD	Final Entry Label
	CPLEVEL	G		DMKCQYCL
	CPUID	G		DMKCQYCP
	FILES	G		DMKCQHFG
	LINKS	G		DMKCQQQL
	LOGMSG	all		DMKCQYLM
	NAMES	all		DMKCQSNA
	PER	all	DMKCMDQG	DMKPEQRY
	PF	G	DMKCMDQP	DMKCQYPF
	PRINTER, PRT	G	DMKCMDQP	DMKCQHTG
	PROCESSR	all		DMKCQPQP
	PUNCH, PCH	G		DMKCQHNG
	READER, RDR	G		DMKCQHRG
	SCREEN	G		DMKCQSSC
	SECUSER	G		DMKCQUSE
	SET	G		DMKCQUST
	SPMODE	G,O		DMKCQYSP
	S370E	G		DMKCQYS3
	TERMINAL	G		DMKCQUTE
	TIME	G		DMKCQYTI
	USERS	all		DMKCQYUS
	USERID	G		DMKCQYUI
	VIRTUAL	G		DMKCQGQV
	VMSAVE	G		DMKCQYVS
	CPTRAP	P	DMKCMDQQ	DMKCQCPT
	ALL	G	DMKCMDQV	DMKCQGQA
	CHANNELS	G		DMKCQGQH
	CONSOLE	G		DMKCQGQC
	DASD	G		DMKCQGQD
	GRAF	G		DMKCQGQG
	LINES	G		DMKCQGQL
	STORAGE	G		DMKCQGQS
	TAPES	G		DMKCQGQT
	UR	G		DMKCQGQU
SET	AFFINITY	O	DMKCMDSO	DMKCFYAS
	CPASSIST	O		DMKCFOSC
	FAVORED	O		DMKCSOSF
	JOURNAL	O		DMKJRLSE
	PRIORITY	O		DMKCFOSP
	QDROP	O		DMKCFOSQ
	RESERVE	O		DMKCFOSR

Command	**Subcommand**	**Function Type**	**Entry Label in DMKCMD**	**Final Entry Label**
	SASSIST	O		DMKCFOSA
	S370E	O		DMKCFOS3
	DUMP	R	DMKCMDSR	DMKCFUDU
	LOGMSG	R		DMKCFULO
	MITIME	R		DMKCFUMI
	PAGING	A	DMKCMDSA	DMKCFUPA
	SRM	A		DMKSRMSS
	MODE	C	DMKCMDSC	DMKCFUMO
	RECORD	C		DMKCFURE
	ACNT	G	DMKCMDSG	DMKCFSAC
	AFFINITY	G		DMKCFYAG
	ASSIST	G		DMKCFYSA
	AUTOPOLL	G		DMKCFSAP
	CONCEAL	G		DMKCFYSC
	CPCONIO	G		DMKCFSCC
	CPUID	G		DMKCFSCP
	ECMODE	G		DMKCFSEC
	EMSG	G		DMKCFSEM
	IMSG	G		DMKCFSIM
	ISAM	G		DMKCFSIS
	LINEDIT	G		DMKCFSLE
	MIH	G		DMKCFVMI
	MSG	G		DMKCFSMG
	NOTRANS	G		DMKCFSNT
	PAGEX	G		DMKCFSPX
	RUN	G		DMKCFSRN
	SMSG	G		DMKCFSSM
	STBYPASS	G		DMKCFVSB
	STMULTI	G		DMKCFVSM
	SVCACCL	G		DMKCFSSA
	TIMER	G		DMKCFYSM
	VMCONIO	G		DMKCFSVC
	VMSAVE	G		DMKCFSVS
	WNG	G		DMKCFSWG
	370E	G		DMKCFS37

Chapter 10. Function Codes

Figure 33 indicates the DIAGNOSE codes used in VM/SP and gives a brief explanation of their uses.

Note: ALL in the Class column of the figure stands for all IBM-defined privilege classes, except class ANY.

Function Code	Class	Function	DMKHVC Label	DMKHVD Label	DMKHVE Label	DMKHVF Label
000	all	Store extended identification code.		HVDSTIDX		
004	C,E	Examine data from real storage.		READCPC		
008	all	Execute VM/SP CP commands.	HVCONFN			
00C	all	Pseudo-timer facility.	HVCHRON			
010	all	Release virtual storage pages.	HVCPGRL			
014	all	Manipulate input spool files.		HCDSPRD		
018	all	Standard DASD I/O.	HVCDISK			
01C	F	Clear error recording area.		HVDLRER		
020	all	General virtual I/O.	HVCFAKE			
024	all	Virtual device type inquiry.		HVDDTYP		
028	all	Dynamic CCW modification.	HVCDCPM			
02C	C,E,F	Get DASD address of error recording and number of cylinders allocated for error recording.			HVEEREP1	
030	C,E,F	Read a page of error recording data.			HVEEREP2	
034	C,E	Read the system dump spool file.		HVDRSDF		

Figure 33 (Part 1 of 3). Function Codes for DIAGNOSE Instruction

Function Code	Class	Function	DMKHVC Label	DMKHVD Label	DMKHVE Label	DMKHVF Label
038	C,E	Read the system symbol table.		HVDRDSYM		
03C	A,B,C	Dynamically updates the VM/SP directory.		HVDDIRCT		
040	all	Clean up after virtual IPL by device.		HVDIPL		
044		Reserved for IBM use.				
048	all	Notify first level CP that this is a second level VM/370 or VM/SP system and this virtual machine has issued SVC 76.	HVCEXIT			
04C	all	Generate accounting records.		HVDACCT		
050	A,B,C	Saves 3704/3705 control program image.		HVD3705		
054	all	Enable or disable external interruptions.		HVDEXPA		
058	all	Virtual console interface for 3270.	HVCGRAF			
05C	all	Edit message according to EMSG settings.	HVCEMSG			
060	all	Provide virtual machine storage size.	HVCSTOR			
064	all	Load, find, or purge a named system.	HVCSYS			
068	all	Virtual Machine Communication Facility.	HVCVMCF			
06C	all	Low-address-protection interface for shadow table maintenance.	HVCSTBY			
070	all	Virtual machine accounting interface.		HNDVMAI		
074	A,B,C	Loads a 3800 printer named system into virtual storage.		HVD3800		
078	all	MSS communication.	HVCSSS			
07C	all	Virtual RDEVBLOK creation.				
080	all	Processes MSSFCALL instruction.	HVCMSSF			

Figure 33 (Part 2 of 3). Function Codes for DIAGNOSE Instruction

Function Code	Class	Function	DMKHVC Label	DMKHVD Label	DMKHVE Label	DMKHVF Label
084	B	Update-in-place a VM/SP directory control statement in its online control block form.	DMKUDU			
088		Reserved for IBM use.				
08C	all	Accesses certain device-dependent information.			HVEQRLY	
094	all	Dumps virtual storage.			HVEDUMP	
098	all	Real I/O support for virtual machines.			HVEDRIO	
0A0	all	Retrieve a group name.	HVCGRP			
0B0	all	Access diagnostic information saved for protected application facility users.			HVEPROT	
0B4	all	Virtual printer external attribute buffer manipulation.			HVEXABD	
0B8	all	Spool file external attribute buffer manipulation.			HVEXABS	
0BC	all	Opens a spool file for a spooled reader device.				HVFOSPID
0C8	all	Set a language for CP.				HVFNLSST
0CC	E	Save a CP message repository for a language.				HVFNLSSA
0D0	all	3480 tape volume serial support. users.	HVCVOL			
0D4	B	Specify an alternate userid.				HVFALTID
0FC		Reserved for IBM use.				
100		Start of functions specified by a user.	HVCUSER			

Figure 33 (Part 3 of 3). Function Codes for DIAGNOSE Instruction

Appendix A. Extended Control-Program Support

Extended Control-Program Support

Extended Control-Program Support for VM/370 (ECPS:VM/370) is also applicable to system with VM/SP. ECPS:VM/370 consists of three hardware-assisted parts:

- Control program assist (CP assist) - defines new hardware instructions to assist CP routines and functions or, in two cases, as new interpretations of existing VM/SP instructions. CP assist does not operate in a VM/SP system that runs under VM/SP.

- Expanded virtual machine assist - provides an expansion of the existing virtual machine assist.

- Virtual interval timer assist - provides a more accurate hardware updating of the interval timer for the virtual machine.

See *VM/SP Planning Guide and Reference* for a list of the processors on which ECPS is available.

ECPS Interaction with Other Functions

- Virtual machine assist - The expanded virtual machine assist can be enabled only if virtual machine assist is also enabled.

- Program event recording - No PER events are recognized by CP assist. Virtual machine assist does recognize PER events for certain instructions. PER events are not recognized during the updating of the virtual interval timer.

- VS1 assist - ECPS:VM/370 and VS1 assist do not interfere with each other.

- DOS emulator - If the DOS emulator is active, virtual machine assist is disabled. CP assist and the virtual interval timer assist are not disabled if the DOS emulator is active.

Control By Control Register 6 and MICBLOK Assist Control Field

The contents of control register 6 exercise overall and absolute control over virtual machine assist, CP assist, expanded virtual machine assist, virtual interval timer assist, and virtual machine extended-facility assist. Values in control register 6 share control of functions provided by expanded virtual machine assist with the setting of bits in the MICBLOK's assist control field. The use of the assist control field is described later on under the topic `Expanded Virtual Machine Assist.`

The following table defines the contents of control register 6:

Bit	Description
0	Virtual machine assist enabled if on, disabled if off
1	Virtual machine in problem state if on, in supervisor state if off
2	ISK and SSK instructions not allowed if on, allowed if off
3	System/360 instructions only if on, System/370 instructions if off
4	Virtual SVC interrupts not allowed if on, allowed if off
5	Shadow table fix up allowed if on, not allowed if off
6	Control program assist enabled if on, disabled if off
7	Virtual interval timer support enabled if on, disabled if off
8-28	Real address of virtual machine pointer list
29	Virtual machine extended-facility assist enabled if on, disabled if off
30-31	Unused, must be zero

The following table shows the interaction between virtual machine, extended virtual machine, CP, and virtual interval timer assists.

Virtual Machine Assist	Control Program Assist	Virtual Interval Timer Assist	Enabled Assists	Operator Command	User Command
N	N	N	None	B, D	
N	N	Y	None	*	
N	Y	N	CP assist	B, C	
N	Y	Y	CP assist	*	
Y	N	N	Virtual machine assist	A, D	E, NOTMR
Y	N	Y	Virtual machine assist, virtual interval timer assist	A, D	E, TMR
Y	Y	N	Virtual machine assist, CP assist, expanded virtual machine assist	A, C	E, NOTMR
Y	Y	Y	Virtual machine assist, CP assist, expanded virtual machine assist, virtual interval timer assist	A, C	E, TMR

* Not possible with VM/SP
A indicates SET SASSIST ON
B indicates SET SASSIST OFF
C indicates SET CPASSIST ON
D indicates SET CPASSIST OFF
E indicates SET ASSIST ON with TMR or NOTMR as indicated

Trace Table Entries

The first bit of each VM/SP trace table entry generated by ECPS is set to one. In a 3031 attached processor environment the second bit is set to one when the entry is generated on the attached processor. Information about VM/SP trace table entries is contained in the *VM Diagnosis Guide.*

Relationships between Hardware Assists

Figure 34 on page 420 illustrates the possible ways of running a virtual machine with various combinations of hardware assists and how the SET command affects their operation.

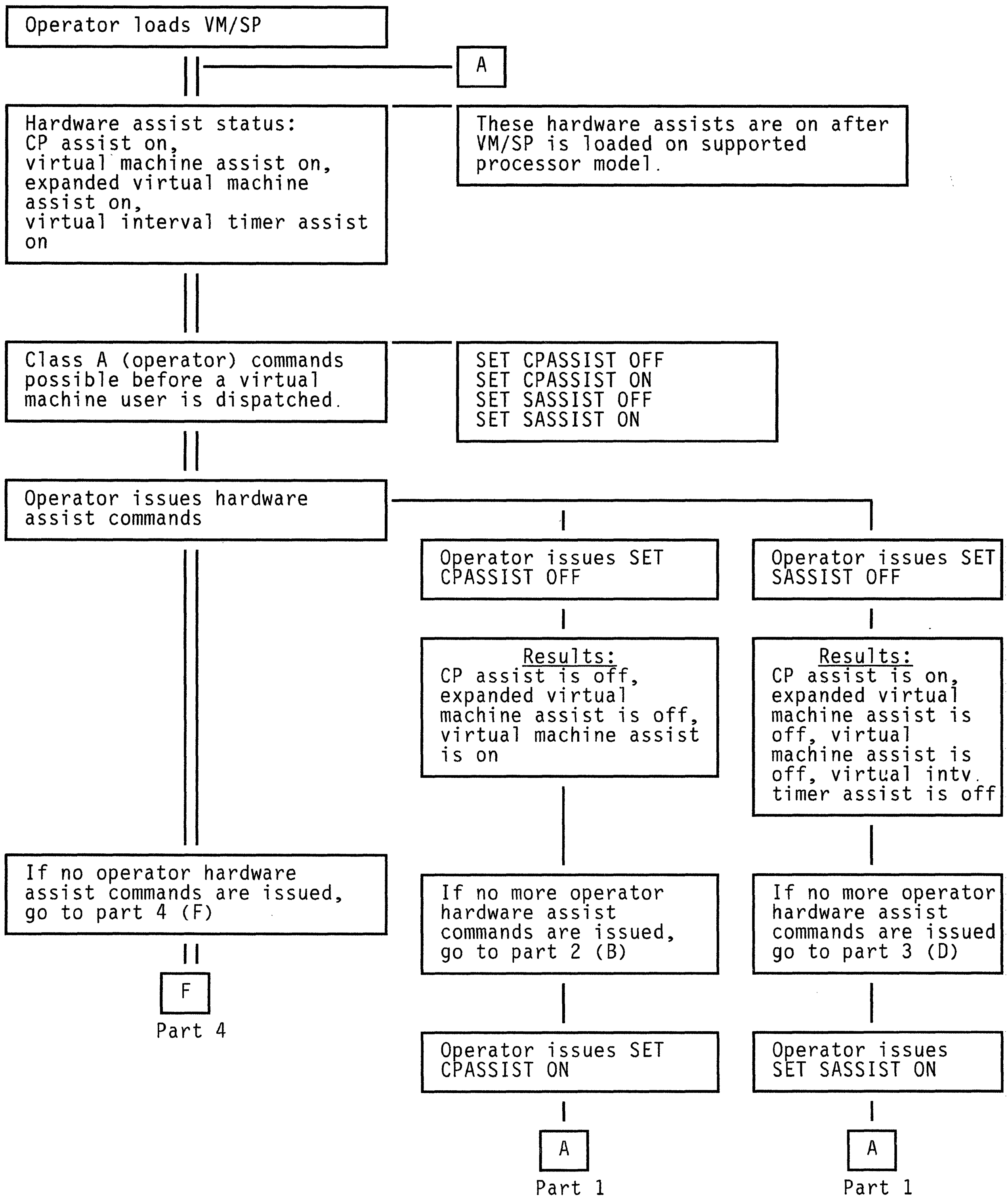

Operator loads VM/SP
A
Hardware assist status: CP assist on, virtual machine assist on, expanded virtual machine assist on, virtual interval timer assist on
These hardware assists are on after VM/SP is loaded on supported processor model.
Class A (operator) commands possible before a virtual machine user is dispatched.
SET CPASSIST OFF
SET CPASSIST ON
SET SASSIST OFF
SET SASSIST ON
Operator issues hardware assist commands
Operator issues SET CPASSIST OFF
Operator issues SET SASSIST OFF
Results: CP assist is off, expanded virtual machine assist is off, virtual machine assist is on
Results: CP assist is on, expanded virtual machine assist is off, virtual machine assist is off, virtual intv. timer assist is off
If no operator hardware assist commands are issued, go to part 4 (F)
If no more operator hardware assist commands are issued, go to part 2 (B)
If no more operator hardware assist commands are issued go to part 3 (D)
F
Part 4
Operator issues SET CPASSIST ON
Operator issues SET SASSIST ON
A
Part 1
A
Part 1

Figure 34 (Part 1 of 4). Hardware Assist Relationships

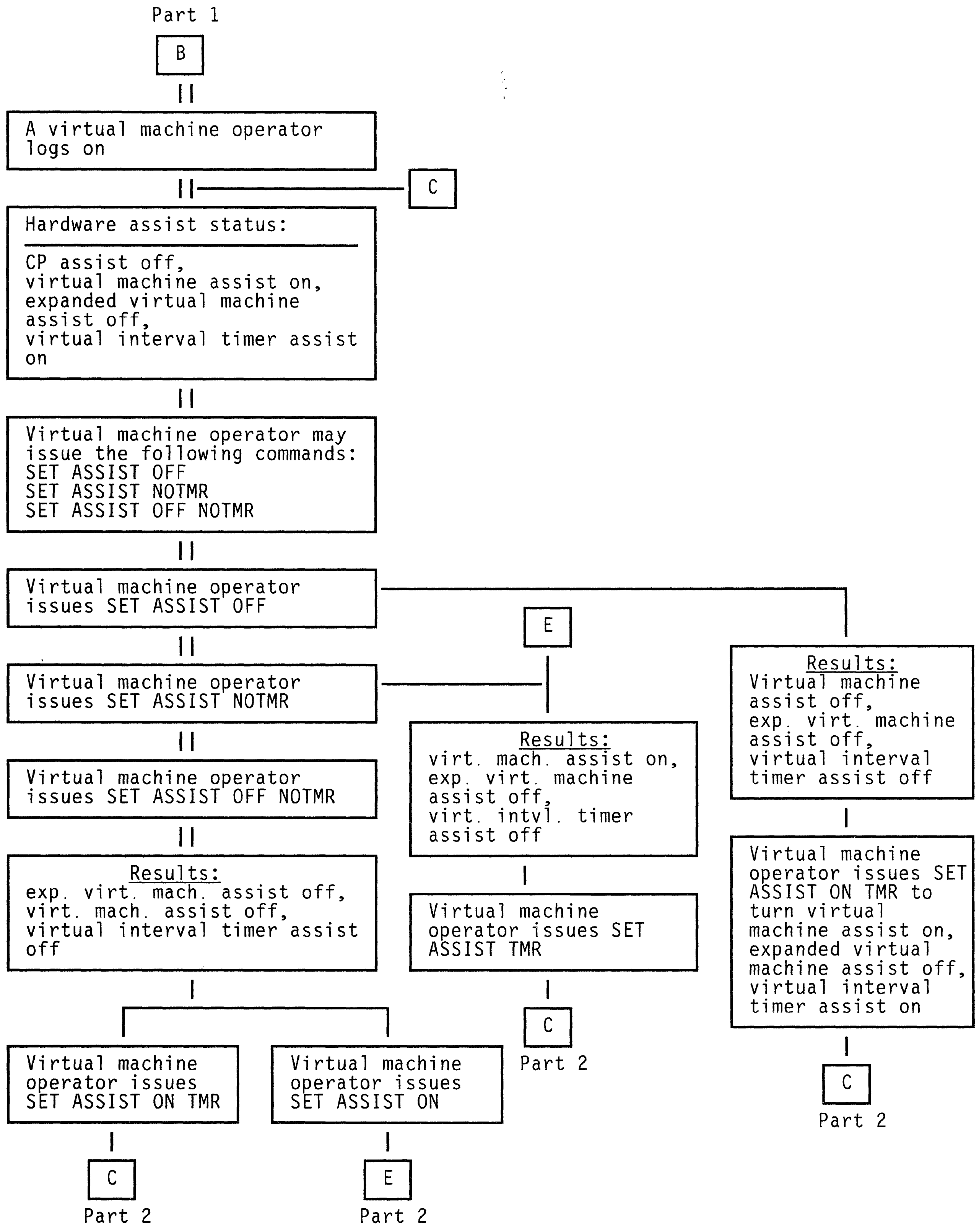

Part 1
B
A virtual machine operator logs on
C
Hardware assist status:
CP assist off,
virtual machine assist on,
expanded virtual machine assist off,
virtual interval timer assist on
Virtual machine operator may issue the following commands:
SET ASSIST OFF
SET ASSIST NOTMR
SET ASSIST OFF NOTMR
Virtual machine operator issues SET ASSIST OFF
Virtual machine operator issues SET ASSIST NOTMR
Virtual machine operator issues SET ASSIST OFF NOTMR
Results:
exp. virt. mach. assist off,
virt. mach. assist off,
virtual interval timer assist off
Virtual machine operator issues SET ASSIST ON TMR
C
Part 2
Virtual machine operator issues SET ASSIST ON
E
Part 2
E
Results:
virt. mach. assist on,
exp. virt. machine assist off,
virt. intvl. timer assist off
Virtual machine operator issues SET ASSIST TMR
C
Part 2
Results:
Virtual machine assist off,
exp. virt. machine assist off,
virtual interval timer assist off
Virtual machine operator issues SET ASSIST ON TMR to turn virtual machine assist on, expanded virtual machine assist off, virtual interval timer assist on
C
Part 2

Figure 34 (Part 2 of 4). Hardware Assist Relationships

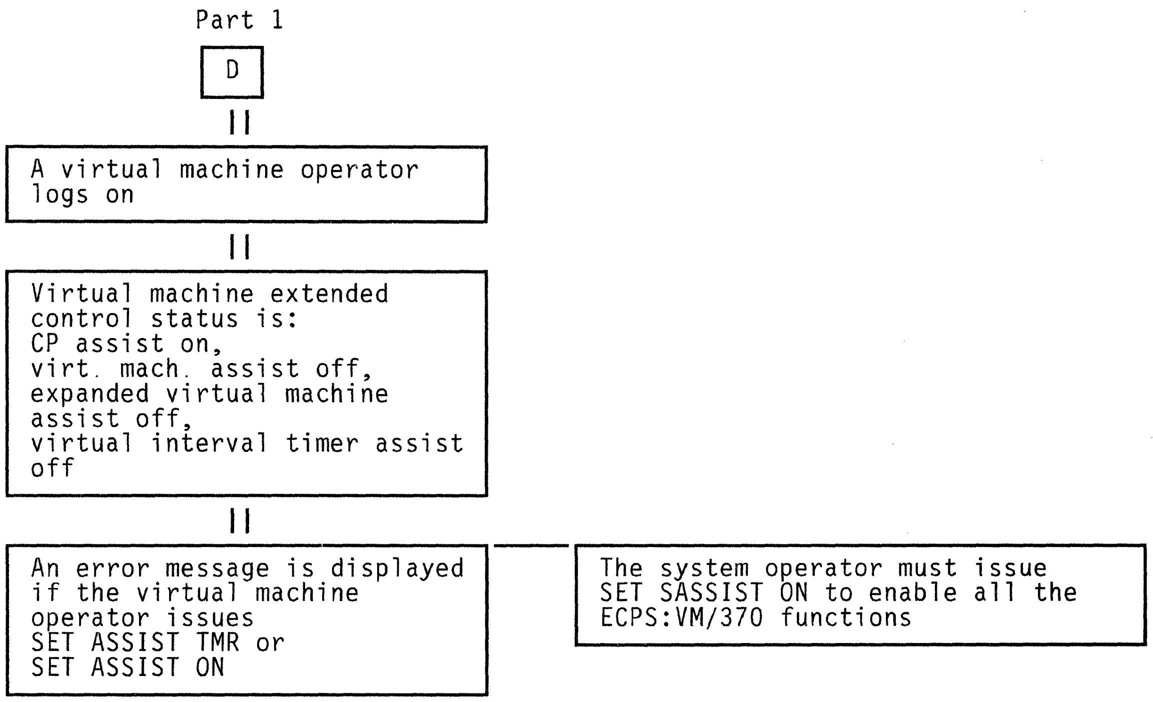

Part 1
D
A virtual machine operator logs on
Virtual machine extended control status is: CP assist on, virt. mach. assist off, expanded virtual machine assist off, virtual interval timer assist off
An error message is displayed if the virtual machine operator issues SET ASSIST TMR or SET ASSIST ON
The system operator must issue SET SASSIST ON to enable all the ECPS:VM/370 functions

Figure 34 (Part 3 of 4). Hardware Assist Relationships

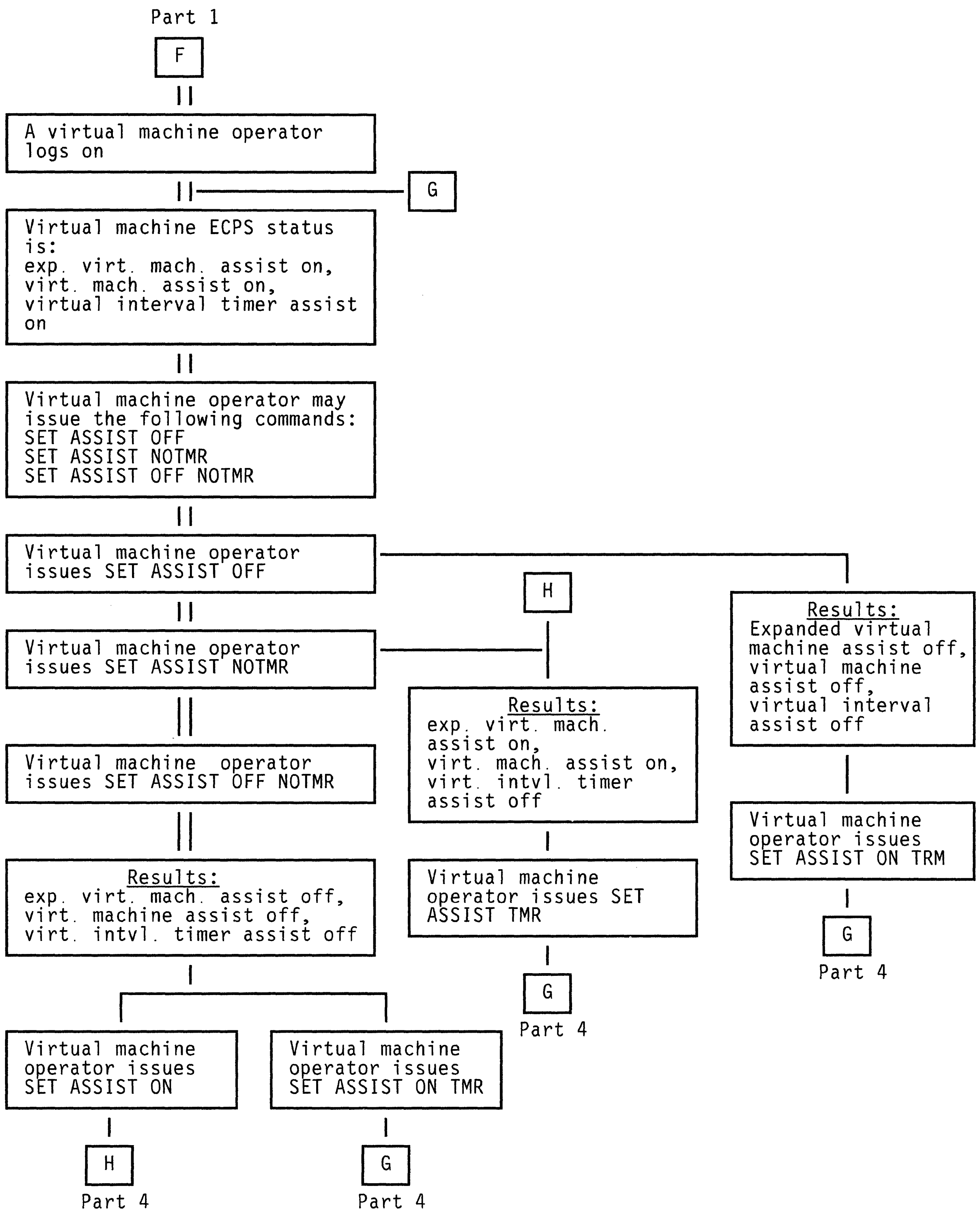

Part 1
F
A virtual machine operator logs on
G
Virtual machine ECPS status is: exp. virt. mach. assist on, virt. mach. assist on, virtual interval timer assist on
Virtual machine operator may issue the following commands: SET ASSIST OFF SET ASSIST NOTMR SET ASSIST OFF NOTMR
Virtual machine operator issues SET ASSIST OFF
Virtual machine operator issues SET ASSIST NOTMR
Virtual machine operator issues SET ASSIST OFF NOTMR
Results: exp. virt. mach. assist off, virt. machine assist off, virt. intvl. timer assist off
Virtual machine operator issues SET ASSIST ON
H
Part 4
Virtual machine operator issues SET ASSIST ON TMR
G
Part 4
H
Results: exp. virt. mach. assist on, virt. mach. assist on, virt. intvl. timer assist off
Virtual machine operator issues SET ASSIST TMR
G
Part 4
Results: Expanded virtual machine assist off, virtual machine assist off, virtual interval assist off
Virtual machine operator issues SET ASSIST ON TRM
G
Part 4

Figure 34 (Part 4 of 4). Hardware Assist Relationships

Appendix B. VM/SP MSS Support

VM/SP MSS Support

Following are annotated flow diagrams for the logic to support the IBM 3850 Mass Storage System.

Logon a User Having a Minidisk on an Unmounted System Volume

DMKLNK, CHK3330V
A required system volume is not mounted, try to get a 3330V mounted if the minidisk is a 3330.

DMKSSSLN
Entry to mount an MSS system volume.

DMKSST, FINDLOOP
Allocate a SYSVIRT real 3330V device. This may involve demounting a volume which is mounted but not in use. If there are none such volumes available, issue message DMKSST070E and return with return code 8.

DMKSSS, BLDCOMMT
Construct an MSSCOM, filling in the volume serial, device address selected, type of request (mount), and userid.

DMKSSS, DEMDUP
Build a CPEXBLOK for the return to DMKLNK after the MSC has processed the request. Chain it from field MSSTASK2. Build a CPEXBLOK for the return to DMKLNK after the mount is complete (pack change interruption received on the 3330V). Chain it from field MSSTASK1.

DMKSSS, SCHMSSC
Put the MSSCOM in the queue, generate an attention interruption for the communication device if necessary, and exit to DMKDSP.

DMKSSV, HVC04ENT
Entry when DIAGNOSE code X'78', subcode 4 is received. OS/VS is ready to process an MSC request. Place the next MSSCOM in the virtual machine, and return to DMKHVC.

DMKSSV, HVC08ENT
Entry from DMKHVC when DIAGNOSE code X'78', subcode 8 is received. The MSC has processed the mount request.

DMKSSV, RESETMQR
If there was an MSS error, write message DMKSSV073E and return to DMKLNK with return code 8.

DMKSSS, CHAINMOR
If there was not an MSS error, indicate that the MSSCOM is now waiting for the pack change interruption. Write message DMKSSS078I. Return to DMKLINK with return code 4.

DMKLNK, MNTSETUP
Return from DMKSSS. Save the current workarea and control information. Return to caller.

DMKDSB
Entry from DMKDAS on pack change interruption. If the device is a 3330V, look for an MSSCOM waiting for this volume serial. If one is found, stack a CPFXBLOK for entry to DMKSSSEN. Exit to DMKDSP.

DMKSSSEN
Pick up the CPEXBLOK for DMKLNKSS and stack it.

DMKLNKSS
Complete the LINK processing for the minidisk.

Logon a User Having a 3330V Dedicated as a 3330V

DMKLOJ, CALLMSSA
Determine that a virtual 3330V is needed, save the UDEVBLOK, call DMKSSSL1.

DMKSSSL1
Go through device allocation, etc., to schedule a mount.

DMKLOJ, MSSMOUNT
If an MSS mount is in process (return code 4 from DMKSSS), proceed to get the next directory statement. Otherwise, find the RDEVBLOK for the device that DMKSSS allocated and continue the dedicate process.

DMKLOMSS
Entry from DMKDSB and DMKSSSEN after mount.

DMKSCNRU
Get the RDEVBLOK

DMKVDSAT
Attach the virtual device.

DMKLOM, TSTV333V
If the virtual device is a 3330V, set flag RDEV333V to indicate that there is no CP MSS CCW prefix.

DMKLOM, FREEUDEV
If there is virtual I/O waiting, as indicated by a CPEXBLOK address in field MSSTASK3 of the MSSCOM used for the mount, stack the IOBLOK. Return to DMKDSP.

Process DIAGNOSE Code X'78'

DMKSSVHV
Entry from DMKHVC when DIAGNOSE code is X'78'.

DMKSSV, HVC00ENT
The entry subcode was 0. Save the communication device address and the communicator VMBLOK address. Set PSAMSS indicating that the MSC is now available.

DMKSSV, HVC04ENT
The entry subcode was 4. If there is an MSSCOM in the queue to be processed, call DMKPTRAN to get the communicator's buffer address. Put the MSSCOM in the virtual machine buffer.

DMKSSV, HVC08ENT
The entry subcode was 8. The MSC has processed a request. If there was an error, write message DMKSSV073E, dequeue the MSSCOM, stack the return to the DMKSSS caller from MSSTASK2 with a return code 8, and return to DMKHVC. If there was no MSC error, stack the MSSTASK2 CPEXBLOK with a return code of 4, and return to DMKHVC.

Generate the Channel Program Prefix for a 3330V

DMKCCW
Entry to generate a real channel program from a virtual machine channel program.

DMKCCW, CCWINDSD
If the real device is a 3330V, set a flag indicating that the MSS channel program prefix is needed.

DMKCCW, CCW02
If the prefix-needed flag is on and the virtual device is not a virtual 3330V, put the prefix in the RCWTASK.

DMKCCW, DASDTBL AND DEDDTBL
These are tables of addresses of routines that are to get control to process specific CCW operation codes for DASD and dedicated devices. In each subroutine, a check is made to see if there is an unresolved MSS prefix. If so, it checks to see if the virtual channel program contains a SEEK. If so, it checks to see if the argument is used to generate the

SEEK argument for the prefix. If not, the prefix CCW is set to SEEK to cylinder 0.

Generate the Channel Program Prefix for CMS I/O to a 3330V

DMKDGD
Entry to process I/O requests to DASD as initiated by the special DIAGNOSE code '78' interface from CMS.

DMKDGD, NOPRE
If the real device is a 3330V, set up the prefix in the RCWTASK.

DMKDGD, CHKMOUNT
The VDEVBLOK for the virtual device could not be found. Check to see if there is an MSS mount in process for the required system volume. If so, build a CPEVBLOK for this request, put the address in the MSSTASK3 field of the MSSCOM, and exit to DMKDSP.

Process a Staging Adapter Cylinder Fault

DMKIOTIN
Entry when ending status is received from a device. Check to see if the CSW contains CE-DE with no error status.

DMKIOT, TESTCYL
If the device type is a 3330V, see if the CE-DE is in the MSS prefix NOP CCW. If not, or if the device is dedicated as a virtual 3330V, stack the IOBLOK.

DMKSSUI2
Set the IOBFLT flag, indicating that a cylinder fault is being resolved. Chain the IOBLOK from the REDEVFIOB field in the RDEVBLOK. Build a TRQBLOK to recognize missing attention interruptions and put it on the timer queue. Exit to the dispatcher.

Process an Attention Interrupt from a 3330V

DMKIOS, IOSUNSOL
Entry to process unsolicited I/O interrupts.

DMKIOS, CALLMSSA
If the interrupt is an attention, the device is a 3330V, and it is not dedicated as a 3330V. Call DMKSSSI1 to restart I/O.

DMKSSUI1
Find each IOBLOK for this device that has the IOBFLT flag set. Find the associated timer queue element and remove it from the timer queue. Turnoff IOBFLT so that the IOBLOK can be restarted when the device is available.

Appendix C. MVS/System Extensions/Product Support

MVS/System Extensions and MVS System Product support, available as a part of VM/SP, allows an MVS system running in a virtual machine to utilize the enhancements available in the MVS/System Extensions Program Product (Program No. 5740-XE1) or the MVS System Product. Included in MVS/System Extensions and MVS/System Product support is the use of the System/370 Extended Facility of the 3031, 3032, 3033, and 3081 processors, or the System/370 Extended Feature of the Model 158 and 168 processors. For additional information on the System/370 Extended Facility or Feature, see the publication *IBM System/370 Extended Facility*, GA22-7072.

Figure 35 illustrates the relationship among MVS/System Extensions, MVS/System Product, VM/SP, and the System/370 Extended Facility.

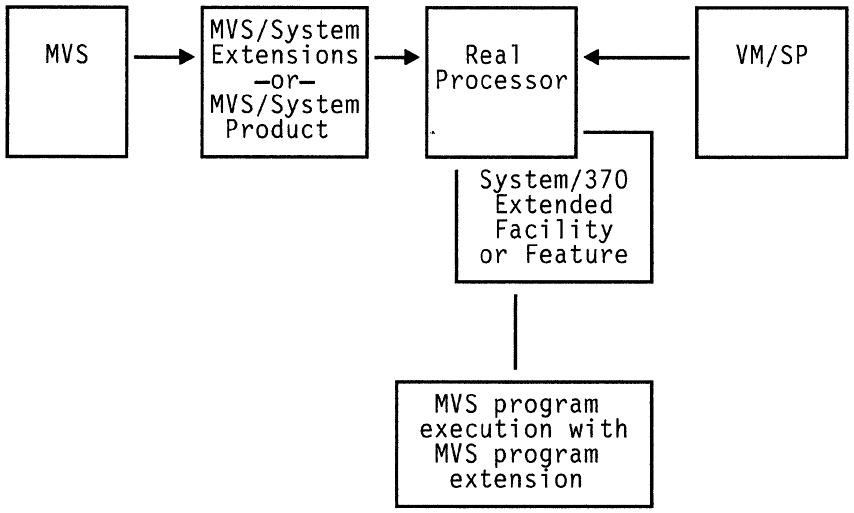

MVS
MVS/System Extensions -or- MVS/System Product
Real Processor
VM/SP
System/370 Extended Facility or Feature
MVS program execution with MVS program extension

Figure 35. Relationship Among MVS/System Extensions or MVS/System Product, VM/SP, and the System/370 Extended Facility

The system operator enables and disables MVS/System Extensions or MVS/System Product support for all virtual machines using the SET S370E ON/OFF command. The user enables and disables it for his own virtual machine via the SET 370E ON/OFF command.

VM/SP determines the status of MVS/System Extensions or MVS/System Product support through two bits in DMKPSA: CP370EAV and CP370EON. CP370EAV indicates whether the feature is available (1=yes, 0=no). CP370EON indicates whether the feature, if available, may be used by virtual machines (1=yes, 0=no). When DMKCPI executes an MVS/System Extensions or MVS/System Product instruction, and the instructions does not cause a program check, DMKCPI sets the value of CP370EAV to one. CP370EAV may not be turned on or off after it is set. CP370EON is initialized to the same value as CP370EAV; it may be turned on or off via the SET S370E command.

MVS/System Extensions and MVS/System Product support includes the following features:

- Low-address protection
- Common-segment facility
- Invalidate Page Table Entry (IPTE) instruction
- Test Protection (TPROT) instruction
- Virtual Machine extended-facility operations.

Low-Address Protection

Low-address protection prevents improper storing by instructions using logical storage addresses between 0 and 511. This facility is designed to prevent inadvertent program destruction of those storage locations that are used by the processors to fetch new PSWs during interruption processing. Low-address protection does not apply to the following:

- The storing of status by the processors (that is, old PSWs, logout data)
- Channel stores (for example, the CSW, LCL, or data)
- Diagnose instructions issued by the virtual machine.

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 between 0 and 511 are permitted. When this bit is zero, stores are permitted; when it is one, stores are not permitted. When an instruction attempts a store using an address between 0 and 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 program interruption for a protection exception occurs.

Low-address protection is in force when all of the following conditions are met:

- Bit 3 of the virtual machine's virtual control register 0 is set to one.

- The VMF370E bit in the VMBLOK is on, indicating that MVS/System Extensions or MVS/System Product functions are enabled for this virtual machine.

- The CP370EON bit in the PSA is on.

- The virtual PSW indicates that EC mode is on.

The storage-protection routine, DMKPSASP, in DMKPSA also checks for violations against low-address protection (this routine is used for simulating privileged instructions, such as Store Then AND System Mask).

Common Segments

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 equals one, 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.

DMKVAT ignores bit 30 of the virtual segment table entry when MVS/System Extensions or MVS/System Product support is enabled (that is, when the VMF370E bit in the VMBLOK and the CP370EON bit in the PSA are both on). If bit 30 equals one, and MVS/System Extensions or MVS/System Product support is not enabled, a translation-specification exception is reflected to the virtual machine.

In DMKVAT, tests are made at labels GIVE012 and LRAEXCP to determine whether the common-segment bit is the only `invalid` segment table entry bit on for a virtual machine with MVS/System Extensions or MVS/System Product support enabled. If it is, and the VMF370E bit in the VMBLOK equals one, and the CP370EON bit of the CPSTAT2 byte of the PSA equals one, then no translation-specification exception is generated. Execution returns to the in-line code after label DMKVATLA (for LRAEXCP) or label DMKVATRN for GIVE012).

Invalidate Page Table Entry (IPTE) Instruction

Execution of the IPTE instruction causes the page table entry designated by the second operand (Ry) to be invalidated in the page table designated by the first operand (Rx), and the associated translation-lookaside buffer (TLB) entries in the system to be purged.

IPTE is an RRE-format instruction (four bytes long) with operation code X'B221'. The Rx field is in bits 24-27 of the instruction and the Ry field is in bits 28-31. The contents of register Rx have the format of a segment table entry with only the page table address used. The contents of register Ry have the format of a virtual address with only the page index used. Other fields of registers Rx and Ry are ignored. The translation format is contained in bits 8-12 of control register 0. The bit positions of register Ry that are selected as a page index depend on the segment and page size specified in control register 0.

Bits 8-12 of Control Register 0	Ry Bit Positions for Page Index
01000 (PS=2K,SS=64K)	16-20
10000 (4K,64K)	16-19
01010 (2K,1M)	12-20
10010 (4K,1M)	12-19

The page table address and page index are used, subject to the rules of dynamic address translation for page table lookup, to form a real address that designates a halfword page table entry. If the page size is 2K, bit 13 of the entry is set to one. Otherwise, the page size is 4K, so bit 12 of the entry is set to one. In any event, key protection applies to the access.

VM/SP simulates an IPTE instruction by setting the proper bit in the virtual machine's page tables and then proceeding as if the instruction were a PTLB (that is, resetting the virtual machine's shadow tables and executing a PTLB with the proper synchronization procedures).

The secondary-operation decode routine for `X'B2xx'` instructions at label BEETWOS causes a branch to the routine DMKPRWIP in module DMKPRW. This routine simulates the IPTE instruction. DMKPRWIP computes the second-level page table (PGT) address, then calls TRANS21 to have the page table entry address translated to a real address. This address is checked for fetch and storage protection via calls to DMKPFAFP and DMKPSASP, respectively. Failing either protection check causes a branch to label PROTEXCP to reflect a protection exception to the virtual machine. After a successful protection check, DMKPRWIF sets the validity bit in the PGTE (PGTPVM) to one, which indicates that the entry is invalid.

If an IPTE instruction is executed by a virtual machine that does not have MVS/System Extensions or MVS/System Product support enabled, an operation exception is returned to the virtual machine.

Test Protection (TPROT) Instruction

The TPROT instruction is an SSE-format instruction (six bytes long) with operation code `X'E501'`. The location specified by the first-operand address is tested for protection (including low-address protection) exceptions, using the key specified in bits 24-27 of the second-operand address. The first-operand address is subject to translation when dynamic address translation is on. Condition codes are set as follows:

```
CC=0  Both fetching and storing are permitted.
CC=1  Fetching is permitted, but storing is not.
CC=2  Neither fetching nor storing is permitted.
CC=3  Translation not available.
```

VM/SP simulates the TPROT instruction in subroutine DMKPRWTP by comparing the key specified by bits 24-27 of the second-operand address to the key for the 2048-byte block specified by the swap table entry corresponding to the first-operand address. The virtual condition code is set to reflect the results of the comparison.

Virtual Machine Extended-Facility Assist

The IBM System/370 Extended Facility includes four lock instructions, six tracing instructions, and the Fix Page and SVC Assist instructions, all of which are MVS-dependent instructions. All are SSE-format instructions (six bytes long) with operation code X'E5xx', where xx represents the secondary-operation code, which can take values between X'02' and X'0D' These instructions are described in more detail in *IBM System/370 Extended Facility*, GA22-7072.

MVS/System Extensions and MVS/System Product support includes the virtual machine extended-facility assist for the 12 MVS-dependent operations. Control of this facility is handled by bits 1 and 29 of control register 6. Bit 1 is the virtual-machine problem-state bit (one indicates the problem state, and zero indicates the supervisor state), and bit 29 activates the virtual machine extended-facility assist for the MVS-dependent instructions. If bit 29 is one and bit 1 is zero, the MVS-dependent operations are executed, even when bit 15 of the real PSW is zero.

If bit 1 is one or bit 29 is zero, then all of the MVS-dependent operations cause program interruptions (privileged operation exceptions).

VM/SP does not support the simulation of the MVS-dependent instructions; these instructions are executed by the hardware only.

Virtual machine extended-facility assist is activated in the following way:

When the VMF370E bit in the VMBLOK is one, bit 1 of real control register 6 is set to the same value as bit 15 (the problem state bit) in the virtual machine's PSW (VMPSW). Then the following tests are made:

- Is the global MVS/System Extensions or MVS/System Product status bit equal to one (bit CP370EON in byte CPSTAT2 of the PSA)?

- Is the extended-control-mode bit equal to one (bit 12 of VMPSW)?

If these tests are met, then:

- Bit 29 of real control register 6 is set to one (this activates the virtual machine extended-facility assist).

- Bit 3 of real control register 0 is set to the value in virtual control register 0 (this indicates whether low-address protection is enabled).

If these tests are not met, then:

- Bit 29 of real control register 6 is set to zero.
- Bit 3 of real control register 0 is set to zero.

These settings deactivate virtual machine extended-facility assist and low-address protection, respectively. In either case, processing continues normally.

Summary of Changes

To obtain editions of this publication that pertain to earlier releases of VM/SP, you must order using the pseudo-number assigned to the respective edition. For:

- Release 4, order LT00-1603
- Release 3, order LT00-1366
- Release 2, order LQ60-0892
- Release 1, order LT60-0892.

Summary of Changes for LY20-0892-3 for VM/SP Release 5

Transparent Services Access Facility (TSAF)

Transparent Services Access Facility is a facility that lets users connect to and communicate with local or remote virtual machines within a group of systems. The Transparent Services Facility consists of the TSAF virtual machine component, APPC/VM, and two CP system services. APPC/VM is a modified subset of IUCV. With the TSAF virtual machine, it provides services within a single system and throughout a group of systems, unlike IUCV, which provides services only within a single system. The TSAF virtual machine component handles communication between systems by letting APPC/VM paths span more than one system. The QUERY CPTRAP command has also been added.

New Modules Generated:

DMKCQC, DMKIDR, DMKIUB,
DMKIUN, DMKIUP, DMKIUS,
DMKTRX, DMKCRM

National Languages Supported by VM/SP

VM/SP now supports the following national languages: American English, German, French, Kanji, and Uppercase English. Updates have been made to modules and data areas providing this support, specifically, those handling CP messages.

New Modules Generated: DMKHVF, DMKMES, DMKVBM

Alternate Nucleus Support Enhancement

Alternate Nucleus support makes it easier to create and IPL backup copies of the CP nucleus when the primary nucleus is damaged or unavailable.

Printer Support Enhancements

The Printer Support Enhancements include the addition of a SPOOL System Service facility which provides support for a printer subsystem. The DESTination option allows you to select a specific printer or punch to process your print, punch, or console file. Two new DIAGNOSE codes allow a user to specify additional information about a print file. The CMS PRINT command has been enhanced to support an OVersize option and a special carriage control character to allow a longer data line.

LOGON/LOGOFF Enhancements

The LOGON/LOGOFF enhancements improve system availability to users and resolve the problem of conflicting messages during LOGOFF processing.

New Module Generated: DMKUSQ

Miscellaneous

- A new diagnose code, DIAGNOSE code X'D4' provides support for an alternate userid.
- The Error Logging System Service, a new CP system service, allows a virtual machine to receive a copy of all records currently written to the VM/SP CP Error Recording Area.
- An enhancement to SPOOL File Compression Support improves the reliability of transmitting spooled data between systems.
- Added support for access verification routines provides a standard interface to the RACF/VM 1.7.1 or user-written routines that can provide a higher level of security.
- Enhancements made to support of ASCII devices.
- Minor editorial and technical changes have been made throughout this publication.

Summary of Changes for LY20-0892-2 for VM/SP Release 4

Group Control System (GCS)

GCS is a new component of VM/SP for Release 4. It is a virtual machine supervisor. Like CMS, GCS depends on CP for reliability and availability. However, unlike CMS, GCS provides a multitasking environment. (Whereas CMS supports an execution environment for only one task at a time.) Included with this is DIAGNOSE Code X'98', which provides real I/O support for virtual machines.

New Module Generated: DMKVMG

User Class Restructure (UCR)

UCR allows an installation to specify up to 32 privilege classes for CP commands.

New Modules Generated: DMKCMD, DMKOVR

Shared/Nonshared Restriction

With the addition of this support, any attempt to construct a virtual device configuration that would mix SHARED and NONSHARED device types on the same virtual control unit is rejected.

CP FRET Trap

The CP FRET Trap can be used as an aid in solving problems caused by improper use of CP free storage and to solve many storage overlay problems.

DMKFRE/DMKFRT split

Module DMKFRE is split into modules DMKFRE and DMKFRT. DMKFRE handles all requests for free storage as well as calls to DMKFRET to release free storage. If the call to DMKFRET cannot be handled by the microcoded CP Assist FRET function, the DMKFRTT entry in DMKFRT handles the request.

New Module Generated: DMKFRT

VMDUMP Enhancements

DIAGNOSE function X'94' is available to allow a virtual machine to request dumping of its virtual storage. Also, the three address range restriction has been removed from the VMDUMP command.

New Module Generated: DMKVME

Control Program (CP) Initialization Enhancements

CP initialization modules DMKCPI and DMKCPJ are been split into eight separate modules, and DMKCKP is been split into six modules. Included in this change is a restructure of the logic and thorough documentation of all the modules. This will make CP easier to service and maintain, and help to solve addressability problems. Also, a new option, REIPL, has been added to the SHUTDOWN command; when specified, this causes an automatic warm start to follow the system shutdown.

New Modules Generated:

DMKALO, DMKCKD, DMKCKF, DMKCKH,
DMKCKM, DMKCKN, DMKIDU, DMKMNT,
DMKOPE, DMKSEG, DMKTOD.

IUCV Enhancements

The Inter-User Communications Vehicle (IUCV) has been enhanced to support data movement from discontiguous buffers on the SEND, RECEIVE, and REPLY functions. The IUCV macro has been updated to handle a "BUFLIST=" parameter on the SEND and RECEIVE functions, and a "ANSLIST =" parameter on the SEND and REPLY functions.

New Modules Generated: DMKVCQ, DMKVCS

Terminal Enhancements

The DIAL command is now supported for SNA devices.

New Module Generated: DMKRGE

Stand-Alone Dump

The Stand-Alone Dump facility is an enhancement to VM/SP serviceability. It provides the support personnel with capability of dumping up to 16 megabytes of real storage. It is required to dump real storage when VM/SP cannot create a CP abend dump.

New Module Generated: DMKSAD

CPTRAP Enhancements

CPTRAP is a major service aid used in problem determination. Enhancements to the CPTRAP command provide two additional functions (GROUPID and WRAP) and one additional record type (X'3D'). Additions to the CPTRAP reduction routine will help make it easier to review the trap data by providing the ability to display and print formatted output of the data in the CPTRAP file.

New Module Generated: DMKTRU

3480 Magnetic Tape Subsystem Support

The 3480 is a buffered magnetic subsystem consisting of one control unit which can address up to 8 drives, or two control units which can each address up to 16 drives.

New Module Generated: DMKTPE

Input/Output Configuration Program (IOCP)

Changes in the IOCP code are reflected in this manual.

Other Module Splits:

Module(s) Being Split	Resulting Modules
DMKCPU	DMKCPM and DMKCPU
DMKCSO	DMKCSF and DMKCSO
DMKCQP	DMKCQP and DMKCQT
DMKEMA, DMKEMB, and DMKEMC	DMKEMA, DMKEMB, DMKEMC, DMKEMD, and DMKEME
DMKIOS	DMKIOQ and DMKIOS
DMKLOG	DMKLOG and DMKLOJ
DMKLOH	DMKLOH and DMKLOM
DMKSST	DMKSST and DMKSSV
DMKTAP	DMKTAP and DMKTAQ
DMKVCA	DMKVCA and DMKVCB

MISCELLANEOUS

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

Integration of Functional Enhancements Applied to Release 3

VM/SP provides support for the following hardware devices:

- 3800 Printing Subsystem Model 3 Compatibility

  New Modules Generated:

  DMKRSF (split from DMKRSE)
  DMKSEQ (split from DMKSEP)
  DMKVSX (split from DMKVSP)

- 3290 Information Panel
- 4248 Printer
- 3370 Direct Access Storage Models A2 and B2

  New Module Generated: DMKVDB

- 4361/4381 Processors.

Glossary of Terms and Abbreviations

This section explains or defines the terms, acronyms, and abbreviations that appear in this manual. For a complete list of terms used in VM/SP refer to the *VM/SP Library Guide, Glossary, and Master Index*, GC19-6207. You may also want to refer to the *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems.*, GC20-1699.

A

abnormal termination. The cessation of processing prior to planned termination.

Advanced Program-to-Program Communication/VM (APPC/VM). An Application Program Interface (API) for communicating between two virtual machines that is mappable to the SNA LU 6.2 APPC interface and is based on IUCV functions. Along with the TSAF virtual machine, APPC/VM provides this communication within a single system and throughout a group of systems.

annotated flow diagram. A diagram consisting of logic labels and commentary. It describes the general flow and use of CP logic modules and their relationships to other modules while performing a specific function or task.

AP. See attached processor.

APPC/VM. See Advanced Program-to-Program Communication/VM.

attached processor (AP). A processor with no I/O capability. An attached processor is always linked to the processor initialized for I/O handling.

C

CC. Condition code.

CCH. Channel check handler.

CCS. See Console Communication Service.

CCT. Communication control table.

CCW. See channel command word.

channel command word (CCW). A double word at the location in main storage specified by the channel address word. One or more CCWs make up the channel program that directs data channel operations.

channel status word (CSW). An area in storage that provides information about the termination of input/output operations.

CMS. See Conversational Monitor System.

Console Communication Service (CCS). A CP component which helps process information (that is ultimately heading to or from a SNA terminal screen) as it passes between CP and VSCS (VTAM SNA Console Support). This helps a SNA terminal serve as an operator's virtual console.

Control Program (CP). The component of VM/SP that manages the resources of a single computer so that multiple computing systems appear to exist.

Conversational Monitor System (CMS). A virtual machine operating system that provides general interactive time sharing, problem solving, and program development capabilities, and operates under the control of CP.

CPU. Central Processing Unit.

CSW. See channel status word.

D

DASD. See direct access storage device.

DAT. See dynamic address translation.

DDR. DASD dump restore.

deadline priority. A number that is used to sort a virtual machine into the eligible lists and the run list. It is based upon an estimate of when the virtual machine should complete its next queue slice.

dispatch list. In VM/370, a list of virtual machines that are to receive a time slice on the processing unit. The virtual machine currently executing is called the runuser. When virtual machines are dropped from the dispatch list, replacement is made from the eligible list. See also eligible list.

direct access storage device (DASD). A storage device in which the access time is effectively independent of the location of the data.

dynamic address translation (DAT). In System/370 virtual storage systems, the change of a virtual address to a real storage address during execution of an instruction.

E

EBCDIC. See extended binary-coded decimal interchange code.

ECC. Error checking and correction.

ECPS. Extended Control-Program Support.

ECSW. Extended channel status word.

eligible list. In VM/370, a list of virtual machines that are potentially executable, but are not placed on the dispatch list to compete for processing unit resources because of the current load on the system. See also dispatch list.

EOF. End of file.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 characters, each represented by eight bits.

F

FB-512. Refers to the IBM 3370 and 3310 Direct Access Storage Device.

FCB. Forms control buffer.

FIFO (first-in-first-out). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time.

flush list. A set of pages available to replenish the free list.

free list. A list maintained by CP that points to a set of pages that can be allocated to satisfy both virtual machine and system page requests.

I

IPL. Initial Program Load.

Inter-User Communication Vehicle (IUCV). A communications facility that allows users to pass information. It enables a program running in a virtual machine to communicate with other virtual machines, with a CP system service, and with itself.

ISAM. Indexed sequential access method.

ISK. Insert storage key.

IOCDS. I/O configuration data set.

IOS. I/O Supervisor.

I/O. Input/Output.

IUCV. See Inter-User Communication Vehicle.

L

LIFO (last-in-first-out). A queuing technique in which the next item to be retrieved is the item most recently placed in the queue.

M

MCH. Machine check handler.

module. A file whose external references have been resolved.

MSS. 3850 Mass Storage System.

MSSF. Monitoring and service support facility.

Multiple Virtual Storage (MVS). An alternative name for OS/VS2 release 2.

MVS. See Multiple Virtual Storage.

O

OS. operating system.

P

page. A 4K byte-long block of data that has a virtual address and can be transferred between real storage and virtual storage.

page frame. An area of real storage that can store a page.

page table. A table in CP that indicates whether a page is in real storage where frequently used pages are paged out. It provides high speed paging.

PCI. Program controlled interruption.

PER. Program event recording.

program status word (PSW). An area in storage used to indicate the order in which instructions are executed, and to hold and indicate the status of the computer system. Synonymous with processor status word.

PSA. Prefix storage area.

PSW. See program status word.

Q

queue 1. In VM/370, a queue of interactive users from which the dispatcher selects users for the dispatch list.

queue 2. In VM/370, a queue of noninteractive users from which the dispatcher selects users for the dispatch list.

queue 3. In VM/370, this is an extension of queue 2 scheduling. It helps to distinguish between noninteractive virtual machines and those that are frequently switching back and forth between queue 2 and queue 1.

R

Remote Spooling Communication Subsystem networking (RSCS). A program product for VM/SP, it is a special-purpose subsystem that sends and receives messages, files, commands, and jobs over a computer network.

RMS. Recovery Management Support

RSCS. See Remote Spooling Communications Subsystem networking.

S

segment table. A table used in dynamic address translation to control user access to virtual storage segments. Each entry indicates the length, location, and availability of a corresponding page table.

shared segment. A 64K segment of storage within a saved segment or a discontiguous saved segment. Read-only and reentrant code in the segment can be shared by multiple virtual machines.

SIO. Start I/O.

SNA. Systems Network Architecture.

SPF. Storage protect feature.

spool, spooled, spooling. Relates to the reading of input data streams and the writing of output data streams on auxiliary storage devices.

SVC. Supervisor Call instruction.

T

Transparent Services Access Facility (TSAF). A component of VM/SP or VM/SP HPO that handles communications between systems by letting APPC/VM paths span more than one system.

TSAF. See Transparent Services Access Facility.

V

virtual address. An address that refers to virtual storage or a virtual I/O device address. It must, therefore, be translated into a real storage or I/O device address when it is used.

virtual machine. A functional equivalent of a real machine.

Virtual Machine Communication Facility (VMCF). A CP function that provides a method of communication and data transfer between virtual machines operating under the same VM/SP systems.

VMCF. See Virtual Machine Communication Facility.

VM/SP (Virtual Machine/System Product). Refers to the VM/SP program package when you use it in conjunction with VM/370 Release 6.

virtual=real option. A VM/370 performance option that allows a virtual machine to run in VM/370's virtual=real area. This option eliminates CP paging and optionally, CCW translation for the virtual machine.

virtual storage. Storage space that can be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computing system and by the amount of auxiliary storage available, and not by the actual number of main storage locations.

VM/VS handshaking feature. A communication interface between VM/370 and OS/VS1 that makes each system control program aware of certain capabilities and requirements of the other.

VM/370. IBM Virtual Machine Facility/370.

VSCS. See VTAM SNA Console Support component.

VTAM SNA Console Support component (VSCS). A VTAM application which allows a SNA terminal user to logon to VM/SP though the Group Control System (GCS) facility.

Numerics

2305. Refers to IBM 2305 Fixed Head Storage, Models 1 and 2.

270X. Refers to IBM 2701, 2702, and 2703 Transmission Control Units or the Integrated Communications Adapter (ICA) on the System/370 Model 135.

2741. Refers to the IBM 2741 and the 3767, unless otherwise specified.

3081. Refers to the IBM 3081 Processor Unit model D16.

3088. Refers to the IBM 3088 Multisystem Communications Unit, Models 1 and 2.

3262. Refers to the IBM 3262 Printer, Models 1 and 11.

3270. Refers to a series of display devices, namely the IBM 3275, 3276, 3277, 3278, 3279 Display Stations, and the 3290 Information Panel. A specific device type is used only when a distinction is required between device types.

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 3330/3333 Model 1 or 3330/3333 Model 11 compatibility mode.

3340. Refers to the IBM 3340 Disk Storage, Models A2, B1, and B2, and the 3344 Direct Access Storage Model B2.

3350. Refers to the IBM 3350 Direct Access Storage Models A2 and B2 in native mode.

3370. Refers to the IBM Direct Access Storage Models A1, A2, B1, and B2.

3375. Refers to the IBM 3375 Direct Access Storage.

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.

3422. Refers to the IBM Magnetic Tape Subsystem.

3480. Refers to the IBM 3480 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.

37XX. Refers to the IBM 3704, 3705, and 3725 Communication Controllers.

3800. Refers to the IBM 3800 Printing Subsystems, Models 1, 3, and 8. A specific device type is used only when a distinction is required between device types. References to the 3800 Printing Subsystem Model 3 apply to both Models 3 and 8 unless otherwise explicitly stated. The IBM 3800 printing Subsystem Model 8 is available only in selected world trade countries.

3880. Refers to the IBM 3880 Storage Control Units.

4245. Refers to the IBM 4245 Line Printer.

4248. Refers to the IBM 4248 Printer.

Bibliography

Here is a list of IBM books that can help you use your system. If you do not see the book you want in this list, you might want to check *IBM System/370, 30XX, and 4300 Processors Bibliography*, GC20-0001.

Prerequisite Publication

IBM General Information Binary Synchronous Communications, GA27-3004

IBM System/360 Principles of Operation, GA22-6821

IBM System/370 Principles of Operation, GA22-7000

IBM 3270 Information Display System Components Description, GA27-2749

Virtual Machine/System Product (VM/SP):

Introduction, GC19-6200
CMS for System Programming, SC24-5286
CP Command Reference, SC19-6211
CP for System Programming, SC24-5285
Operator's Guide, SC19-6202

Virtual Machine (VM):

CP Internal Trace Table (Poster), LX24-5202
Diagnosis Guide, SC24-5241
System Facilities for Programming, SC24-5288

Corequisite Publication

Virtual Machine/System Product (VM/SP):

Data Areas and Control Block Logic Volume 1 (CP), LY24-5220
Data Areas and Control Block Logic Volume 2 (CMS), LY24-5221
Installation Guide, LY24-5237
OLTSEP and Error Recording Guide, SC19-6205
Planning Guide and Reference, SC19-6201
System Messages and Codes, SC19-6204
Service Routines Program Logic, LY20-0890
Transparent Services Access Facility Reference, SC24-5287

Virtual Machine (VM):

Running Guest Operating Systems, GC19-6212

In addition, for EREP processing the following OS/VS Library publications are required:

IBM OS/VS, DOS/VSE, VM/370 Environmental Recording Editing and Printing (EREP) Program, GC28-0772

IBM OS/VS, DOS/VSE, VM/370 Environmental Recording Editing and Printing (EREP) Program Logic, SY28-0773.

If the IBM 3850 Mass Storage System is attached, the following publications are required:

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

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

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

If the VTAM SNA Console Support (VM/VSCS) product is used, the following publication is a prerequisite:

IBM VM/VSCS Diagnostics, LY38-3033.

Notes:

1. *References in text to titles of publications are given in abbreviated form.*
2. *The VM/SP Library Guide, Glossary, and Master Index, GC19-6207 describes all the VM/SP books and contains an expanded glossary and master index to all the books in the VM/SP library.*

The VM/SP Library (Part 1 of 3)

Evaluation

General Information	GC20-1838
Introduction	GC19-6200

Index

Library Guide, Glossary, and Master Index	GC19-6207

Planning

Planning Guide and Reference	SC19-6201
Running Guest Operating Systems	GC19-6212
Release 5 Guide	SC24-5290
Distributed Data Processing Guide	SC24-5241

Installation

Installation Guide	SC24-5237

Applications

Application Development Guide	SC24-5247
Programmer's Guide to the SRPI for VM/SP	SC24-5291

Operation

Operator's Guide	SC19-6202

Reference Summaries

To order all of the Reference Summaries, use order number SBOF-3242

Commands (General User)	SX20-4401
Commands (Other than General User)	SX20-4402
SP Editor Command Reference Summary	SX24-5122
EXEC 2 Reference Summary	SX24-5124
Sys.Prod Interpreter Reference Summary	SX24-5126
CMS Primer Summary of Commands	SX24-5151
CMS Primer Line-Oriented Summary of Commands	SX24-5159
Problem Reporting Summary (Poster)	SX24-5171
Summary of End Use Tasks and Commands (Poster)	SX24-5173

The VM/SP Library (Part 2 of 3)

End Use

- Terminal Reference GC19-6206
- CMS Primer SC24-5236
- CMS Primer for Line-Oriented Terminals SC24-5242
- CMS User's Guide SC19-6210
- CMS Command Reference SC19-6209
- CMS Macros and Functions Reference SC24-5284
- System Product Editor User's Guide SC24-5220
- System Product Editor Command and Macro Reference SC24-5221
- System Product Interpreter User's Guide SC24-5238
- System Product Interpreter Reference SC24-5239
- EXEC 2 Reference SC24-5219
- CP Command Reference SC19-6211
- Quick Reference SX20-4400

Diagnosis

- System Messages and Codes SC19-6204
- System Messages Cross-Reference SC24-5264
- Service Routines Program Logic LY20-0890
- Problem Reporting Guide SC24-5282
- VM Diagnosis Guide LY24-5241
- GCS Diagnosis Reference LY24-5239
- Problem Determination Vol. 1 (CP) LY20-0892
- Data Areas and Control Blocks Vol. 1 (CP) LY24-5220
- Problem Determination Vol. 2 (CMS) LY20-0893
- Data Areas and Control Blocks Vol. 2 (CMS) LY24-5221
- OLTSEP and Error Recording Guide SC19-6205
- VM Problem Determination Reference Information LX23-0347
- VM CP Internal Trace Table (Poster) LX24-5202

The VM/SP Library (Part 3 of 3)

Administration

VM System Facilities for Programming	SC24-5288
CP for System Programming	SC24-5285
CMS for System Programming	SC24-5286
TSAF Reference	SC24-5287
GCS Command and Macro Reference	SC24-5250

Auxiliary Communication Support

VTAM Installation and Resource Definition	SC23-0111
VTAM Customization	SC23-0112
VTAM Operation	SC23-0113
VTAM Messages and Codes	SC23-0114
VTAM Reference Summary	SC23-0135
VTAM Programming	SC23-0115
VTAM Diagnosis Guide	SC23-0116
VTAM Diagnosis Reference	LY30-5582
VTAM Data Areas (VM)	LY30-5583
RSCS Networking Version 2 General Information	GH24-5055
RSCS Networking Version 2 Planning and Installation	SH24-5057
RSCS Networking Version 2 Operation and Use	SH24-5058
RSCS Networking Version 2 Diagnosis Reference	LY24-5228
RSCS Networking Version 2 Ref. Summary	SX24-5135
VM/Pass-Through Facility General Information	GC24-5206
VM/Pass-Through Facility Guide and Reference	SC24-5208
VM/Pass-Through Facility Logic	LY24-5208

Index

International Business
Machines Corporation
P.O. Box 6
Endicott, New York 13760

File No. S370/4300-36
Printed in U.S.A.

LY20-0892-4

VM/SP System Logic and Problem Determination Guide
Volume 1 (CP)
Order No. LY20-0892-4

READER'S COMMENT FORM

Is there anything you especially like or dislike about this book? Feel free to comment on specific errors or omissions, accuracy, organization, or completeness of this book.

If you use this form to comment on the online HELP facility, please copy the top line of the HELP screen.

__________ __________ ______ Help Information line ___ of ___

Note: Do not use this form to report system problems or to request copies of publications. Instead, contact your IBM representative or the IBM branch office serving you.

Would you like a reply? ___YES ___NO

Please print your name, company name, and address:

____

____

____

____

IBM Branch Office serving you: ____________________________

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us.

Reader's Comment Form

CUT
OR
FOLD
ALONG
LINE

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
DEPARTMENT G60
PO BOX 6
ENDICOTT NY 13760-9987

Fold and tape Please Do Not Staple Fold and tape

IBM®

International Business
Machines Corporation
P.O. Box 6
Endicott, New York 13760

File No. S370/4300-36
Printed in U.S.A.

LY20-0892-4

LY20-0892-04